High-Performance Cluster-Scalable Computational Methods for Genomics Applications

Ahmad, T.

DOI
10.4233/uuid:e0eb1883-47c2-402b-b736-4f8e00ebb45f

Publication date
2022

Document Version
Final published version

Citation (APA)
Ahmad, T. (2022). *High-Performance Cluster-Scalable Computational Methods for Genomics Applications*. [Dissertation (TU Delft), Delft University of Technology]. https://doi.org/10.4233/uuid:e0eb1883-47c2-402b-b736-4f8e00ebb45f

High-Performance Cluster-Scalable Computational Methods for Genomics Applications

High-Performance Cluster-Scalable Computational Methods for Genomics Applications

Dissertation

for the purpose of obtaining the degree of doctorate
at Delft University of Technology,
by the authority of the Rector Magnificus prof.dr.ir. T.H.J.J. van der Hagen,
chair of the Board for Doctorates,
to be defended publicly on
Monday 4 July 2022 at 17:30 o'clock

by

Tanveer AHMAD

Master of Science in Computer Engineering,
University of Engineering and Technology, Lahore, Pakistan
born in Karor Lal-Eason (Layyah), Pakistan.

This dissertation has been approved by the promotors.

promotor: Prof.dr. H.P. Hofstee
promotor: Dr.ir. Z. Al-Ars

Composition of the doctoral committee:

Rector Magnificus,	Chairman
Prof.dr. H.P. Hofstee,	Delft University of Technology / IBM, USA
Dr.ir. Z. Al-Ars,	Delft University of Technology
Independent members:	
Prof.dr. K.G. Langendoen	Delft University of Technology
Prof.dr. M.J.T. Reinders	Delft University of Technology
Prof.dr. K.T. Heljanko	University of Helsinki, Finland
Dr. F.Y. Liu	Oak Ridge National Laboratory, USA
Dr. J. Gracia	High Performance Computing Center Stuttgart, Germany

Keywords: Genomics, Variant Calling, Apache Arrow, Apache Spark, MPI

Printed by: Ridderprint

Front & Back: Tanveer Ahmad

ISBN 000-00-0000-000-0

An electronic version of this dissertation is available at
https://repository.tudelft.nl/

"When our time is up, I think we will all regret the time that was wasted, lost, in senseless arguments, in trivial distractions, in the grip of addictions, in pursuing the valueless. We will regret not doing life in a way that leaves us having tasted its beauty & depth."

Farheen

Contents

Summary

The ever increasing pace of advancements in sequencing technologies has enabled rapid DNA/genome sequencing to become much more accessible. In particular, next (second) and third generation sequencing technologies offer high throughput, massively parallel and cost-effective sequencing solutions. Individual sample sequencing data volumes as well as the number of assembled genomes are also growing quickly. These advances in high-throughput sequencing technologies and demand for fast computational processing and downstream analysis of sequencing data in clinical settings is widening the gap between the time spent in sample collection and sequencing versus computational analysis. At the same time, due to the physical limitations induced by the underlying CMOS technology, single processor core performance is no longer improving at the rate of increasing transistors count. In contrast to the traditional performance increases enabled by Moore's law and Dennard's scaling, modern processors performance increases today are coming primarily from adding more cores per processor, instead of increases in the maximum achievable frequency and single-core performance. To improve the scalability and performance optimizations of genome variant calling analysis workflows on modern computing systems, in this dissertation four potential research directions have been selected for further exploration.

The GATK best practices variant calling pipelines often incur huge performance bottlenecks due to repeated storage access in intermediate applications and from inefficient multi-threading performance. First, to exploit the performance of modern processors hardware features like multi-core and vector units on the GATK best practices variant calling pipelines, we introduce ArrowSAM, a columnar in-memory data format to place and process genomics data in-memory thus removing the need for repeated file storage accesses in intermediate variant calling pipeline applications. This format gives better performance and parallel processing capability resulting in reduction in the overall runtime by 4.85x and 4.76x for the WGS and WES datasets, respectively.

In terms of big data scalability, a number of cluster scaled variant calling workflows have been developed using Apache Spark as a scheduling and execution engine. However, the Java environment in an Apache Spark cluster process data on a row-by-row basis, making runtime vectorized execution of data not possible with this execution model. Our second contribution focuses on integration of the Apache Arrow based columnar in-memory data format in the PySpark API to enable exploiting the benefits of vectorized operations in the Python language using user-defined functions on Spark dataframes. This approach outperforms the state-of-the-art by more than 2x for the pre-processing stages.

Arrow Flight, a submodule in the Apache Arrow project, offers high performance, secure, parallel and cross-platform language support for bulk data (Apache Arrow in-memory columnar format) transfers across the network. For our third research contribution, we tested and benchmarked both the scalability and performance of Arrow Flight for client-server as well as cluster scaled communication. For Apache Arrow data, it achieves more than 92% of the maximum possible interconnect throughput. We describe a variant calling workflow on HPC clusters using the Slurm workload manager and Arrow Flight as the data communication framework. This architecture outperforms Apache Spark based workflows achieving both less communication overhead (more than 20-60% depending on cluster size) and better scalability resulting in a 2x speedup.

For our final research contribution reported in this dissertation, we implemented an orthogonal approach that is even more scalable than Apache Spark and Arrow Flight based solutions and offers flexibility to use many different variant callers. We implemented MPI in both BWA-MEM and Minimap2 aligners and created a highly scalable variant calling workflow using existing pre-processing applications accessing a chromosome-based queue that uses MPI-RMA atomic operations. Our approach allows the flexibility to use any variant caller and we have tested DeepVariant, Octopus, GATK HaplotypeCaller and Clair3 variant callers. The resulting approach provides higher performance, better scalability and ease-of-use with reproducibility and accuracy. This approach is 20% faster than other MPI-based implementations for BWA-MEM, and provides better scalability for the rest of the variant calling workflows.

Samenvatting

Door de steeds snellere vooruitgang in sequencing-technologieën is snelle DNA-/genoom-sequencing veel toegankelijker geworden. Met name de next (tweede) en derde generatie sequencing-technologieën bieden hoge doorvoer, massaal parallelle en kosteneffectieve sequencing-oplossingen. Individuele sample sequencing datavolumes en het aantal geassembleerde genomen groeien ook snel. Deze vooruitgang in high-throughput sequencing-technologieën en de vraag naar snelle computationele verwerking en downstream-analyse van sequencing-gegevens in klinische omgevingen vergroot de kloof tussen de tijd die wordt besteed aan het verzamelen van monsters en sequencing versus computationele analyse. Tegelijkertijd, vanwege de fysieke beperkingen die worden veroorzaakt door de onderliggende CMOS-technologie, verbeteren de prestaties van de enkele processor niet langer in het tempo waarin het aantal transistors toeneemt. In tegenstelling tot de traditionele prestatieverbeteringen die mogelijk worden gemaakt door de wetten van Moore en van Dennard, komen de prestatieverbeteringen van moderne processors tegenwoordig voort uit het toevoegen van meer cores per processor, in plaats van verhogingen van de maximaal haalbare frequentie en single-core prestaties. Om de schaalbaarheid en prestatie-optimalisaties van workflows voor het aanroepen van genoomvarianten op moderne computersystemen te verbeteren, zijn in dit proefschrift vier mogelijke onderzoeksrichtingen geselecteerd.

De GATK best-practice pipeline voor het bepalen van genomische varianten, leidt vaak tot enorme prestatieknelpunten vanwege herhaalde toegang tot opslag tussen de componenten van de pipeline en tot inefficiënte multi-threading-prestaties. Ten eerste hebben we, om de prestaties van moderne processor-hardwarefuncties zoals multi-core en vectoreenheden op de GATK best-practice-variant voor het aanroepen van pijplijnen te benutten, ArrowSAM geïntroduceerd, een kolom-georienteerd in-memory gegevensformaat om genomics-gegevens in het geheugen te plaatsen en te verwerken door het verwijderen van de behoefte aan herhaalde toegang tot file-gebaseerde storage tussen de componenten van de pipeline. Dit formaat biedt betere prestaties en parallelle verwerkingscapaciteit, wat resulteert in een vermindering van de totale runtime met respectievelijk 4,85x en 4,76x voor de WGS- en WES-datasets.

Wat de schaalbaarheid van big data betreft, zijn er een aantal clustergeschaalde variant-calling workflows ontwikkeld met Apache Spark voor scheduling- en uitvoering De Java-omgeving in een Apache Spark-cluster verwerkt gegevens echter rij voor rij, waardoor runtime-gevectoriseerde uitvoering van gegevens niet mogelijk is met dit uitvoeringsmodel. Onze tweede bijdrage richt zich op de integratie van het op Apache Arrow gebaseerde, kolomvormige in-memory dataformaat in de PySpark API om de voordelen van gevectoriseerde bewerkingen in de Python-taal te kunnen benutten met behulp van door de gebruiker gedefinieerde functies op Spark-dataframes. Deze aanpak overtreft de state-of-the-art met meer dan 2x voor de pre-processing fasen.

Arrow Flight, een submodule in het Apache Arrow-project, biedt hoogwaardige, veilige, parallelle en platformonafhankelijke taalondersteuning voor bulkgegevensoverdracht (Apache Arrow in-memory kolomformaat) over het netwerk. Voor onze derde onderzoeksbijdrage hebben we zowel de schaalbaarheid en prestaties van Arrow Flight voor client-server als clustergeschaalde communicatie getest en gebenchmarkt. Voor Apache Arrow-gegevens bereikt het meer dan 92% van de maximaal mogelijke interconnect-doorvoer. We beschrijven een variant die de workflow op HPC-clusters aanroept met behulp van de Slurm-workloadmanager en Arrow Flight als datacommunicatieraamwerk. Deze architectuur presteert beter dan op Apache Spark gebaseerde workflows en biedt zowel minder communicatieoverhead (meer dan 20-60% afhankelijk van de clustergrootte) als een betere schaalbaarheid, wat resulteert in een 2x betere snelheid.

Voor onze laatste onderzoeksbijdrage die in dit proefschrift wordt gerapporteerd, hebben we een orthogonale benadering geïmplementeerd die zelfs meer schaalbaar is dan op Apache Spark en Arrow Flight gebaseerde oplossingen en die flexibiliteit biedt om veel verschillende variant callers te gebruiken. We implementeerden MPI in zowel BWA-MEM- als Minimap2-aligners en creëerden een zeer schaalbare variant van de oproepworkflow met behulp van bestaande pre-processing-applicaties die toegang hebben tot een op choromosome gebaseerde queue die MPI-RMA-atomaire bewerkingen gebruikt. Onze aanpak biedt de flexibiliteit om een willekeurige variant caller te gebruiken en we hebben DeepVariant en Octopus, GATK HaplotypeCaller en Clair3 variant bellers getest. De resulterende aanpak biedt hogere prestaties, betere schaalbaarheid en gebruiksgemak met reproduceerbaarheid en nauwkeurigheid. Deze aanpak is 20% sneller dan andere MPI-gebaseerde implementaties voor BWA-MEM, en biedt een betere schaalbaarheid voor variant-calling workflows.

1

Introduction

Prelude: *Deoxyribonucleic acid (DNA) is the chemical name of the molecule that contains the biological instructions that defines the characteristics of all living organisms. DNA is a complex molecular structure represented by four chemical bases (A-C-G-T) which contains all the information and instructions needed to create, maintain and reproduce an organism throughout its life. Since the initial discovery of DNA structure, efforts have been made to understand the exact patterns of nucleotides (bases) in a DNA fragment, called DNA sequencing. In the last five decades, many different sequencing strategies and improvements have been made. The most prominent methods include early Maxam-Gilbert sequencing, Chain-termination (Sanger sequencing), Shotgun sequencing and most recently high-throughput sequencing (HTS) methods including next-generation sequencing (NGS) for "short-reads" and third-generation sequencing for "long-reads". Genome sequencing is changing the whole spectrum of our understanding of health, disease, diagnosis and treatment. Rapid advancements in sequencing technologies enabled development of targeted therapies and personalized medicine. However, sequencing machines produce raw base-pair data, which require further computational methods including quality-control, alignment, assembly and variant calling to enable observing and visualizing variants of concern. Continuous computational algorithmic development and improvements to analyze sequencing data is an ongoing research field. New high performance and power efficient computing infrastructures are necessary to meet the growing data and compute intensive genomics workloads needs.*

In this chapter, we provide a short description of genome sequencing and data processing followed by a brief introduction of computational performance challenges being faced on modern computing systems for processing large amounts of data, particularly genomics datasets, where short time-to-diagnosis is becoming ever more important in a clinical setting. Afterwards, we briefly present some propositions we aim to discuss in this dissertation in the context of high performance and scalable computation methods for genome sequence data processing. Finally, a short outline describing the organization of this dissertation is presented.

The Human Genome Project [1], a first ever effort to sequence all human genes in an individual, announced a final draft sequence of the euchromatic portion of the human genome containing approximately 2.85 billion nucleotides in 2003. A more recent effort to sequence the human genome for some remaining missing repetitive regions completed in 2021 [2]. NGS technologies offer high throughput and cheap sequencing methods like whole genome sequencing (WGS), whole exome sequencing (WES), whole transcriptome - RNA sequencing (RNA-Seq) and single-cell RNA sequencing (scRNA-Seq) for in-depth analysis. These methods can be used to detect genetic variants related to genetic disease or tumors, be used to help identify biomarkers and profiling of whole transcriptome and clustering cell types in different organs and tissue, and be used to find cellular differences and individual cell functions with higher resolution. Generally, sequencing includes the following main steps to extract meaningful (variants) information from the genome: 1. sample collection (blood, saliva, tissue, cell culture etc.), 2. DNA extraction and library preparation from targeted samples, 3. sequencing DNA samples, 4. quality control checks, and 5. raw data processing and downstream data analysis. These steps are shown in Figure 1.1. Our work in this dissertation mainly focuses on the fifth step related to computational processing of raw sequence data generated by sequencers to make this data clean for post-processing analysis. Due to advancements in modern massively-parallel sequencing technologies, high throughput, low cost and immensely high volumes of sequencing data is being produced. Adequate computational power is becoming essential to process this data in a reasonable amount of time. On the other hand, due to technology limitations, single CPU core performance cannot be improved by adding more, smaller and power efficient transistors. Hence to keep continue increasing performance, more CPU cores are being added to the processor. Additionally, new hardware features like vectorization called Single Instruction/Multiple Data (SIMDe) and co-processors/accelerators (FPGA/GPU) are being utilized in/with processors for performance efficiency. Similarly, distributed and parallel high performance computing (HPC) infrastructures are being rigorously explored for these emerging data and compute intensive workloads.

In this context, we first re-design and optimize some of the genomics applications to better utilize modern processor hardware features (multi-core parallelism, vectorized operations, in-memory format to avoid data serialization/deserialization between processes and frameworks, better cache locality exploitation) for efficient and maximum resource utilization. Moreover, we also employ scalable methods for these applications to deploy them on public clouds and HPC clusters for high performance and better scalability.

1.1. Genomics & genome analysis

The *genomics* field emphasizes the understanding of structure, mapping and function of individual genes (the genome) to get insights into their interaction and evolution with respect to one’s environment. Different organisms genomes have different number of base pairs (bps). Each cell in a human body contains a

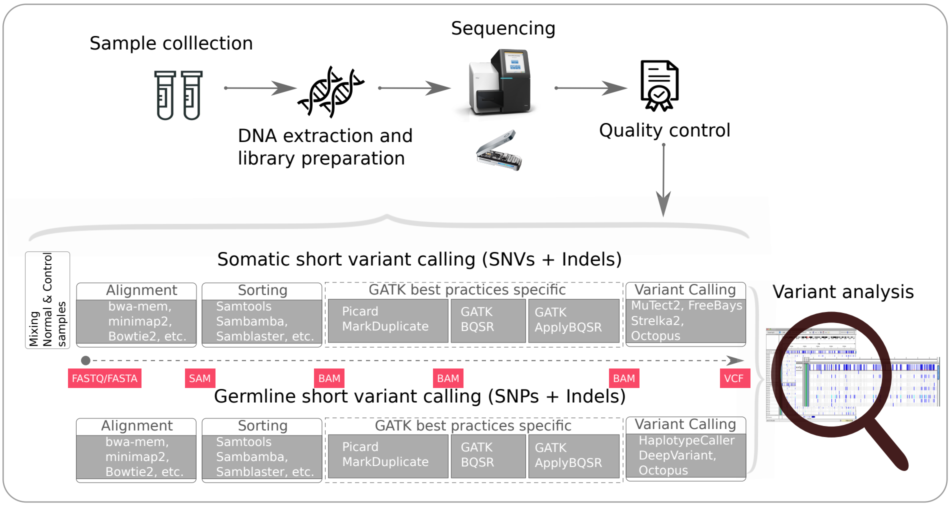


Figure 1.1 An overview of the variant calling process, from sample collection to variant analysis.

complete replication of the approximately 3 billion bps of DNA. DNA is divided into coding regions (or functional units) called genes and non-coding regions which make up the rest of the genome. In many organisms, only a small fraction of the genome consists of coding regions which help encode proteins. The human genome is only composed of roughly 1.5% coding regions while some non-coding regions that do not code proteins may still be used to regulate gene expression. A gene can either result in the creation of a ribonucleic acid (RNA) or a protein resulting from the gene's expression. Gene expression is the most basic function in genetics where genotypes (the complete set of genetic material) gives rise to phenotypes (observable characteristics or traits) in an organism. To understand the breadth and depth of the DNA as whole as well as RNA transcription in an organism, the latest sequencing technologies pave the way froward in both research and clinical application.

Comparative genomics reveal deep insights into nucleotide-level organismal differences for some specific traits among populations from an individual genome sequencing data. These differences are called single nucleotide variants (SNVs), single nucleotide polymorphisms (SNPs) and single base insertions and deletions (InDels), with their corresponding computational method is often called variant calling, which is the focus of this dissertation. A detailed description of somatic short variant calling (SNVs+InDels) and germline short variant calling (SNPs+InDels) is given in Section 2.3.

For both WGS and WES analysis, after performing sequencing on a DNA sample and generating raw sequencing data, typical applied computational variant discovery pipelines include pre-processing, variant calling and variant analysis steps. Pre-processing includes alignment, sorting, GATK best practices specific applications. Afterwards variant calling is performed on pre-processed data followed by the variant analysis as shown in Figure 1.1. Algorithms and applications used in above steps and their corresponding input and output data formats (FASTQ, SAM, BAM and VCF) are also shown in the figure.

In this dissertation, we focus on creating scalable and high performance methods to optimize this variant calling pipeline on both single-node and multi-node computing systems. For single-node computers, we have employed methods to exploit modern hardware features (multi-core, caching, vectorization). We incorporated Apache Arrow (a unified columnar in-memory data format) into existing applications in variant calling pipeline to represent SAM data as columnar in-memory data and called it ArrowSAM [3]. We also used Plasma Object Store for parallelization and inter-process communication (IPC) of data between all the applications in this pipeline. This helped in avoiding intermediate local I/Os in each application. Moreover, using this format, better parallelization is achieved and data cache locality is exploited in all variant calling applications.

In addition to single-node optimizations, we explored multi-node clusters solutions for variant calling. For this purpose first we integrated ArrowSAM into Apache Spark (a unified cluster scaled big data analytics frameworks). The purpose is to replace Apache Spark row-by-row execution with the Apache Arrow columnar data format to exploit the vectorized user-defined functions (UDFs) execution in PySpark and Pandas dataframes. Arrow Flight, a wire-speed data transfer protocol has also been explored for inter-node ArrowSAM data transfer and shuffling to implement variant calling workflows on clusters in Python environment for flexibility and reproducibility purpose. We also used the MPI parallel programming model to speed up different variant calling workflows on HPC clusters. Figure 2.1 shows the overall design of all such implementations.

1.2. Problem description and scope

We outline the following key points providing an overview of the existing state-of-the-art in computing systems, their usage, limitations and future perspective.

- Rapid advances in sequencing technologies enabled producing high-throughput, cost effective and higher coverage sequencing data. This tends to be a new trend in big data paradigms.
- Mainstream usage of genome sequencing for some critical disease diagnostics and personalized medicine will require future computational systems to be as fast and efficient as possible to process the huge amount of genomics data timely.
- Fundamental limitations induced by CMOS technology on the performance and scalability of general-purpose processors in-terms of single-thread performance, maximum achievable frequency and number of cores per processor have opened-up the era of co-processors and accelerators.
- Co-processors and accelerators integration with general-purpose processors is beneficial to some extent for some specific applications but it still

limits the achievable performance scalability for big data applications and particularly genomics data processing at the single node-level.

- Recent developments in high performance computing clusters and distributed cloud computing environments in data centers promise a potential solution for big data processing challenges in the future.
- In conjunction with co-processors and accelerators, the high performance cloud computing infrastructures can provide adequate performance and scalability for genomics data processing.
- Application specific processors designed for genomics processing systems are good candidates for high performance and energy efficient future solutions for this purpose.

The following points more concisely elaborate the problems which this dissertation aims to explore more rigorously in the coming chapters.

- **Problem 1:** Often software programmers and developers focus on the algorithmic aspects of the implementation. For better performance and to reduce power consumption, new processor designs are meant to exploit the different modern architectural characteristics (multi-cores for parallelism, better cache locality, less memory references, vector units usage) of the processors. There exists a gap to map genomics algorithms efficiently on such processors by keeping in-mind these modern processor features.

- **Problem 2:** Usually, local SSD I/O latency is considered almost 1000x more than memory. GATK best practices pipelines recommend multiple applications to accomplish any type of variant calling. All these applications store intermediate output data in SAM/BAM format and consequently each application must read and parse this data again before processing it. This creates overhead of reading/writing to and from I/Os. In some applications the I/O time is more than 50% of the overall runtime.

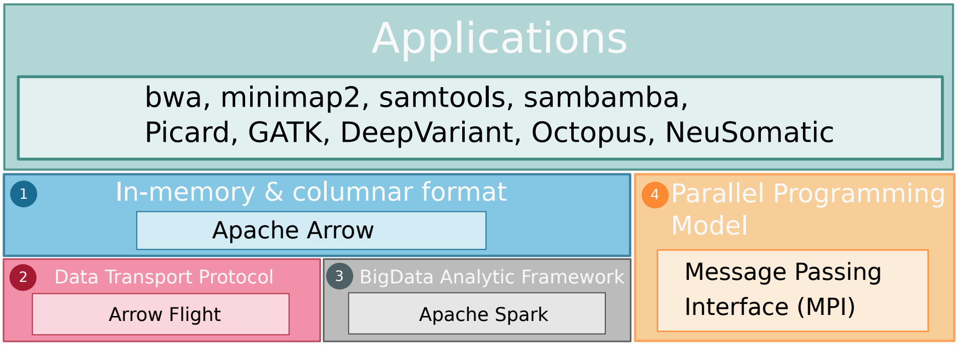

Applications
bwa, minimap2, samtools, sambamba,
Picard, GATK, DeepVariant, Octopus, NeuSomatic
1
In-memory & columnar format
Apache Arrow
4
Parallel Programming Model
Message Passing Interface (MPI)
2
Data Transport Protocol
Arrow Flight
3
BigData Analytic Framework
Apache Spark

Figure 1.2 Representation of stack layers in genomics applications which include the applications, Apache Arrow, Arrow Flight, Apache Spark and MPI.

- **Problem 3:** Apache Arrow in-memory columnar data format supports zero-copy reads for large datasets in inter-process communication without (de)-serialization overheads. This could also help in efficient vectorized data analytics operations and better cache locality. While row based format is preferable in Apache Spark where JVM (Java virtual machine) executors process data on a row-by-row basis. However, runtime vectorized execution of data is not possible with this execution model. Interfacing and using columnar data format (ArrowSAM) on row based execution frameworks (such as Apache Spark) for vectorized execution is challenging but exhibits huge potential performance benefits.

- **Problem 4:** Big data frameworks (Apache Hadoop and Apache Spark) have been extensively used for the scalability (data shuffling and scheduling) of genomics variant calling and other workflows. These frameworks come with some additional computation, dependency and memory overheads by default. Depending on the deployment infrastructure, other scalable, high performance and memory efficient solutions could be explored.

1.3. Research contributions

In the context of the problems described above and their wider implications, we briefly pose a couple of propositions and discuss our research contributions related to them in this dissertation:

Columnar in-memory data formats have already proven helpful in database (transactional) and data-analytic systems.

- **Proposition 1**—Columnar in-memory data formats can also be beneficial in genome variant calling applications for better cache locality, vectorized operations and parallel execution on modern processors.

For this purpose we used GATK best practices variant calling pipeline. First we shortly introduce the different applications in GATK best practices (germline variant calling) pipeline. These applications include: alignment (BWA-MEM/Bowtie2), sorting (Samtools/sambmaba/Picard), duplicate reads removal (Picrd MarkDuplicate), base quality score recalibration (GATK BQSR, ApplyBQSR) and variant calling (GATK HaplotypeCaller). Generally sequence alignment is the first computational stage where we align the raw FASTQ data against a reference genome. BWA-MEM is commonly used algorithm for this purpose. If the output of these tools is stored in columnar format instead of tab delimited text format, we can achieve efficient cross application communication with low overhead. This is done through fast in-memory communication instead of writing intermediate data to I/O storage. Columnar data also speeds up sorting and duplicates removal where we only need to process a subset of the data in SAM files. **Section 3.1** addresses this solution in details.

Furthermore, we also focus on the applications in the GATK best practices pipeline and show that these stages also benefit from columnar and in-memory

SAM data (called ArrowSAM) on which **Section 3.2** expands on our implementation further to achieve better cache locality, less or no additional I/O overhead, multi-cores parallelization and vectorized operations which address the challenges posed by **Problem 1**. This implementation is depicted in ❶ of Figure 5.13.

Columnar in-memory data formats such as Apache Arrow provide an efficient alternative to store and access in-memory data among multiple big data frameworks or applications by avoiding extra (de)-serialization overhead.

- **Proposition 2**—These formats should be integrated into big data frameworks like Apache Spark to avoid (de)-serialization overhead and in-memory data access during data transformation and processing when running different applications inside these frameworks.

Apache Hadoop and Apache Spark big data frameworks have been used extensively to explore the opportunities for the scalability of genomics applications and particularly GATK best practices based variant calling workflow on computing clusters as discussed in the previous contribution. Using Apache Spark just for distributing data and scheduling applications does not exploit the full benefits of Apache Spark in-memory processing. At the same time Spark native computation is row-based. In **Chapter 4** we present the applicability of using Apache Arrow in-memory and columnar data format inside Apache Spark to exploit the benefits of vectorized operations in the Python language. This solution as shown in ❸ of Figure 5.13, utilizes PySpark UDFs and operates on dataframes for both in-memory data storage and shuffling for alignment, sorting and duplicates removal stages. Afterwards DeepVariant (a deep learning based variant caller) has been loosely integrated into the whole workflow to accomplish germline variant calling. As discussed in **Problem 2**, this approach efficiently utilizes maximum system resources with both high performance and linear scalability.

Arrow Flight provides a high performance and parallel interface for bulk data (Apache Arrow in-memory columnar format) transfers across networks. We benchmark for maximum client-server and cluster throughput for this data communications protocol.

- **Proposition 3**—Arrow Flight protocol can achieve high performance data movement throughput for genomics variant calling workflows across computing clusters as compared to big data frameworks such as Apache Spark.

Arrow Flight provides a high performance, secure, parallel and cross-platform language support for bulk data (Apache Arrow in-memory columnar format) transfers across networks. Instead of using big data frameworks (Hadoop and Spark) for data shuffling and transformation on cluster environments as represented in ❷ of Figure 5.13, we employed Arrow Flight as a microservice for data transfer and processing on edge nodes in a local cluster without deserialization of Apache Arrow data. We used SLURM [4] as workload manager on clusters to control the flow and scheduling between variant calling applications as discussed in **Chapter 5**. This approach has huge potential to efficiently address the issues mentioned in **Problem 3** and **Problem 4** with less memory overhead and better linear scalability on clusters.

Genomics variant calling applications like genome alignment and variant calling can be performed on data chunks independently without any data communication or synchronization.

- **Proposition 4**—MPI can be used to achieve bare-metal performance in sequence alignment algorithms and variant calling workflows overall.

MPI promises bare-metal performance for genomics applications on HPC clusters as many genomics applications and algorithms are able to scale up using parallel and distributed processing without the need for synchronization or data sharing. In **Chapter 6** we show the impact of integrating MPI in both short and long reads aligners, BWA-MEM and minimap2 respectively followed by other pre-processing and variant calling applications. The resultant solution provides a highly linear scalable implementation of both aligners as well as variant calling workflows. This solution mainly addresses the concerns described in **Problem 4** and shown in ❹ of Figure 5.13.

1.4. Dissertation outline

The rest of this dissertation is organized as follows. A brief introduction to sequencing technologies and their applications, DNA sequence alignment, assembly and variant calling algorithms and computational methods and infrastructures is presented in Chapter 2. A perspective and future vision for genomics data growth and demanding computer power is also discussed in Section 2.6. In Chapter 3, we mainly focus on node level optimization techniques including in-memory, columnar data format for better cache locality, parallelization. We first discuss how to avoid intermediate local file system I/O accesses for pre-processing (alignment, sorting and duplicate removal) applications in Section 3.1, followed by GATK best practices variant calling workflow in Section 3.2. Apache Spark integration with Apache Arrow to exploit the in-memory columnar SAM format for short reads NGS data germline variant calling workflow is discussed in Chapter 4. In Chapter 5, we first benchmark the different performance aspects of Arrow Flight on a cluster in Section 5.1. This is followed by Section 5.2 where we mainly focus on Arrow Flight microservice integration in-conjunction with the SLURM workload manager on a locally deployed cluster for variant calling workflow scalability. Chapter 6 provides an in-depth description and implementation details of MPI for integration into aligners and variant calling workflows with a thorough comparison with other scalable cluster implementations. Finally, in Chapter 7, we conclude our dissertation contribution with some important highlights followed by a guide with a brief description of future research exploration and directions.

REFERENCES

[1] I. H. G. S. Consortium. "Finishing the euchromatic sequence of the human genome". In: *Nature* 431.7011 (2004), pp. 931–945. issn: 1476-4687. doi: 10.1038/nature03001. url: https://doi.org/10.1038/nature03001.

[2] S. Nurk, S. Koren, A. Rhie, M. Rautiainen, A. V. Bzikadze, A. Mikheenko, M. R. Vollger, N. Altemose, L. Uralsky, A. Gershman, S. Aganezov, S. J. Hoyt, M. Diekhans, G. A. Logsdon, M. Alonge, S. E. Antonarakis, M. Borchers, G. G. Bouffard, S. Y. Brooks, G. V. Caldas, H. Cheng, C.-S. Chin, W. Chow, L. G. de Lima, P. C. Dishuck, R. Durbin, T. Dvorkina, I. T. Fiddes, G. Formenti, R. S. Fulton, A. Fungtammasan, E. Garrison, P. G. Grady, T. A. Graves-Lindsay, I. M. Hall, N. F. Hansen, G. A. Hartley, M. Haukness, K. Howe, M. W. Hunkapiller, C. Jain, M. Jain, E. D. Jarvis, P. Kerpedjiev, M. Kirsche, M. Kolmogorov, J. Korlach, M. Kremitzki, H. Li, V. V. Maduro, T. Marschall, A. M. McCartney, J. McDaniel, D. E. Miller, J. C. Mullikin, E. W. Myers, N. D. Olson, B. Paten, P. Peluso, P. A. Pevzner, D. Porubsky, T. Potapova, E. I. Rogaev, J. A. Rosenfeld, S. L. Salzberg, V. A. Schneider, F. J. Sedlazeck, K. Shafin, C. J. Shew, A. Shumate, Y. Sims, A. F. A. Smit, D. C. Soto, I. Sović, J. M. Storer, A. Streets, B. A. Sullivan, F. Thibaud-Nissen, J. Torrance, J. Wagner, B. P. Walenz, A. Wenger, J. M. D. Wood, C. Xiao, S. M. Yan, A. C. Young, S. Zarate, U. Surti, R. C. McCoy, M. Y. Dennis, I. A. Alexandrov, J. L. Gerton, R. J. O'Neill, W. Timp, J. M. Zook, M. C. Schatz, E. E. Eichler, K. H. Miga, and A. M. Phillippy. "The complete sequence of a human genome". In: *bioRxiv* (2021). doi: 10.1101/2021.05.26.445798. eprint: https://www.biorxiv.org/content/early/2021/05/27/2021.05.26.445798.full.pdf. url: https://www.biorxiv.org/content/early/2021/05/27/2021.05.26.445798.

[3] T. Ahmad, N. Ahmed, J. Peltenburg, and Z. Al-Ars. "ArrowSAM: In-Memory Genomics Data Processing Using Apache Arrow". In: *2020 3rd International Conference on Computer Applications Information Security (ICCAIS)*. 2020, pp. 1–6. doi: 10.1109/ICCAIS48893.2020.9096725.

[4] Slurm. *Slurm workload manager*. 2020. url: https://www.schedmd.com/.

2

Background

Prelude: *This chapter covers background knowledge necessary to understand the context of this dissertation. First, we discuss briefly the genomics field and genome analysis followed by a short description of genome sequencing technologies and their emerging applications. A short introduction of genome DNA sequence alignment, genome assembly and variant calling algorithms is presented. Afterwards, we focus on computing systems, addressing modern processors, hardware accelerators, cloud computing and high performance computing clusters. In addition, we discuss programming models and infrastructures used to run genomics algorithms efficiently. Finally, we conclude this chapter by presenting a broader perspective on advancements in sequencing technologies, growth of data in general and genomics data in particular, while discussing future computing needs for these workloads.*

2.1. Sequencing technologies and applications

To analyze an organism’s DNA for the purpose of understanding and characterizing the unique features it exhibits, the proper order of bases of its DNA should be determined. To identify the DNA sequence, Microarrays are considered a pioneering technology which can detect a relative concentration in a sample of a known DNA sequence.

Further technological advancements led to the birth of Sanger sequencing methods which can detect the order of bases in DNA more accurately. This technique has been deployed in the field for past couple of decades. Sanger sequencing can produce read lengths of approximately 800bp at a time and have been used in multiple research domains from detecting smaller genomic regions in a wide range of samples to variable regions in a genome. The next generation sequencing (NGS) technologies provide higher throughput and lower costs, with their massively parallel sequencing method. The NGS sequencing involves fragmenting DNA/RNA into multiple pieces, adding adapters to the pieces for library preparation, sequencing the libraries, and finally reassembling them to form a genomic sequence. Illumina is a leading manufacturer of NGS technology, which can sequence short-length reads with up to 300 bps (base-pairs). Longer read third generation sequencing technologies are also emerging as a more competitive alternative in terms of cost and throughput with improving accuracy as compared to NGS. They can produce reads of up to hundreds of kilobases (kbps). Of the two leading long reads sequencing manufacturers, Pacific Biosciences (Pacbio) uses the single molecule real time sequencing (SMRT) method, while Oxford Nanopore Technologies (ONT) employs nanoscale pore structures to detect changes in the electrical field surrounding the pore. These technologies promise to make new applications possible such as metagenomics (where very long DNA strands need to be sequenced), as well as personalized medicine (where very high throughput, low cost sequencing is needed).

The advances in DNA sequencing technologies and downstream statistical or machine learning based analysis software are playing a vital role in making and establishing sequencing as a fundamental tool for different types of applications including but not limited to the following:

- to develop targeted molecular therapies using genomic tumor profiling and precision oncology (cancer treatment)
- to detect fetus anomalies in prenatal and perinatal testing
- to find some rare hereditary disorders, genetic variants associated with specific diseases in newborn screening
- diagnosing genetic and hereditary defects and to repair these through genes therapy
- to genetically modify crops for higher nutritional value and disease resilient variants
- to improve the efficiency (production) of livestock and create environmentally resilient breeds

- to study microbiome cultures for viral and infectious disease control

in addition, many more applications in human and community health, agricultural development and animal well-being are being considered.

2.2. Sequence alignment and genome assembly

Sequencing machines usually produce a large amount of raw sequences of a genome. Aligning these sequences (against a reference genome or against each other) by incorporating insertions and/or deletions within the targeted sequence to find an approximate match is called *sequence alignment*. Generally, two types of alignments are used, in global alignment a complete query sequence (end-to-end) is aligned while in local alignment small fragments of large target sequences are matched for similar regions. Pair-wise sequence alignment uses two sequences to find a local or global optimal alignment pattern between them. BWA-MEM [1] and Bowtie2 [2] are widely used short read aligners while Minimap2 [3], Winno [4] and lra [5] are new long read aligners. Similarly stitching small raw DNA fragments/sequences to reconstruct a large consensus/original sequence of DNA through alignment and some heuristics is called *genome assembly*. Shasta [6], Wtdbg2 [7], Flye [8], Hifiasm [9] and Canu [10] are some of the latest long read assemblers. Alignment and assembly are the two most important and fundamental steps for downstream analysis of target genomes.

2.3. Variant calling analysis

Variant calling is indispensable for comparative genomics studies as it reveals deep insights into nucleotide-level organismal differences/variations among populations from an individual's genome sequencing data. Variant calling discerns genetic and/or somatic variations in three categories; single nucleotide polymorphisms (SNPs), insertions and deletions (InDels), and/or structural variants (SVs, may also include copy number variations (CNVs), duplication, translocation, etc).

DNA can mutate in any of the somatic cells (all cells in body except sperm and egg cells) or in germinal or sex cells (eggs and sperm); such variations are referred to as somatic and germline mutations, respectively. A single nucleotide variant *(SNV)* is a substitution of a single nucleotide. SNVs can be rare in one population but common in other populations. *SNPs* are naturally occurring germline variants and are present in $>1\%$ of the population at least. These variants represent a single base (nucleotide) change. *InDels* are single bases which have been inserted, or deleted in a genome when aligning to another reference genome. *Structural variants (SVs)* are observed in an organism's chromosome structures. Generally defined as a region of DNA of approximately 1 kilo base-pairs (kbp) or larger in size having variations (usually greater than 50 bps).One form of SVs is copy number variants (CNVs) also called repeats or duplications which refer to number of copies of a particular gene present in the genome of an individual. DNA sequencing reveals that CNVs are commonly observed in various organisms, particularly in human, which vary from individual to individual. Approximately two thirds of the whole human genome is composed of such repeats.

GATK HaplotypeCaller [11] and FreeBayes [12] are commonly used open-source tools for germline variant calling analysis. Tools like VarScan [13], VarDict [14], MuTect2 [15] are used for somatic variant calling analysis. FreeBayes, SNVer [16] and LoFreq [17] are also used for both germline and somatic variant calling analysis. Pisces [18] and Strelka2 [19] are recently developed open source tools by Illumina for short variant calling to analyze both germline and somatic variations. DeepVariant [20], NanoCaller [21] are deep convolutional neural network (CNN) based germline variant callers, while NeuSomatic [22] is a CNN based somatic variant caller.

Figure 2.1 shows different variant calling workflows that have been implemented in the scope of this dissertation. The first row represents the GATK best practices variant calling pipeline on a single node based on Apache Arrow in-memory and columnar data format. The next three variant calling workflows are cluster scaled (both public clouds and HPC clusters) and use different variant callers other than GATK. These cluster scaled variant calling workflows use combinations of Apache Arrow–Apache Spark–SLURM, Apache Arrow–Arrow Flight–SLURM and MPI–OpenPBS, respectively.

2.4. Computing systems for genome analysis

Bioinformatics and genomics are young research fields and are undergoing enormous growth. Continuously, new algorithms are being developed and released. Heng Li (a leading researcher in the field) recently noted that just in this last year four new short-read mappers (accel-align, chromap, dragmap and snap v2) have appeared and concluded that "Short-read mapping is not solved yet". Due to the fast pace of development of new aligners, developers mainly focus on algorithmic aspects of the solution. At the same time, efforts to improve performance and efficiency are happening rapidly in the context of modern hardware features

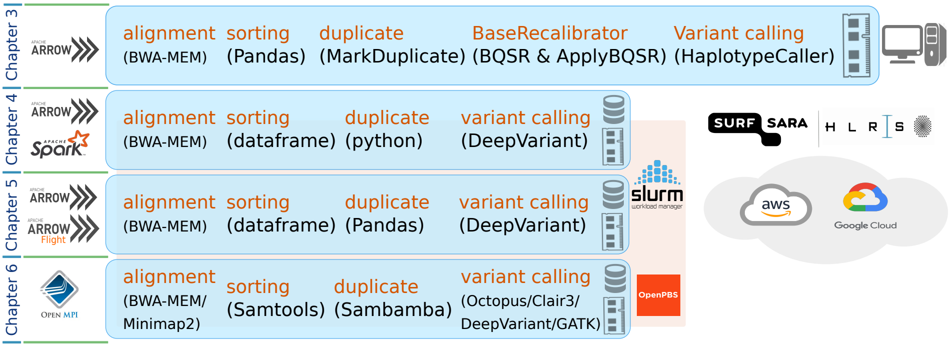


Figure 2.1 Genome variant calling workflows implementation and representation in their respective computing infrastructure explored in this dissertation.

in processors, accelerators usage and scalability of these algorithms. Here, we mention some some of these computing technologies and genomics algorithms developed for them.

2.4.1. Modern processors (Multicores, SIMD)

In the past two decades due to the limitations posed by design complexity, cooling and power consumption, increasing processor frequency was not a viable solution for better instructions per cycle (IPC) per watt performance. Performance improvements now can mainly be achieved by increasing the number of cores on a processor as shown in the microprocessors trend data in Figure 2.2. Both symmetric multiprocessing (SMP, i.e., using multiple identical processors on a single board) and chip-multiprocessing (CMP, i.e., integrated multiple cores inside a single processor), having a shared memory, uniform memory access (UMA) or non-uniform memory access (NUMA) interconnects and techniques have been developed. In addition to increasing the number of cores, processors today implement many powerful and innovative performance improvement techniques. These include scalar vs superscalar processing and in-order vs out-of-order instruction execution pipelines to extract instruction level parallelism (ILP) from sequential programs. Single-instruction multiple-data (SIMD) parallel processing technique (presented in Flynn's taxonomy) was introduced decades ago and being continuously evolved. As in modern workloads matrix multiplication is highly used for image and signal processing applications. BWA-MEM2 [23] and minimap2 [3] extensively and efficiently use multi-threading and SIMD instructions, the former particularly on Intel while the latter also exploits vectorization on both x86 and ARM architectures.

2.4.2. Heterogeneous computing (FPGA, GPU)

In heterogeneous computing systems, normally the CPU works as a host and offloads the highly parallelizable parts of an algorithm on the accelerators. On GPUs and FPGAs these accelerated algorithms are called kernels. Kernels are launched by the host on the device, and kernel data is transferred from host to device (i.e., accelerator) memory. Once data processing on the device is finished, the output is transferred back from device memory to the host again. GPU and FPGA devices are usually connected to PCIe, peripheral to the host. Many programming frameworks have been created for heterogeneous computing, some are cross-vendor and cross-platform like OpenCL, while others are vendor-specific (e.g., oneAPI DPC++ is an Intel specific CPU/GPU/FPGA programming framework). CUDA is used for programming Nvidia GPU kernels. For FPGAs, traditionally Register transfer level (RTL) languages (e.g., VHDL, VerilogHDL) have been used for abstract level digital systems design on FPGAs but due to their high complexity and the resulting longer time-to-market, many FPGA vendors and EDA (electronic design automation) companies have created higher abstraction level programming frameworks in C/C++ called high level-synthesis (HLS) like Intel Quartus, Xilinx Vitis and Catapult.

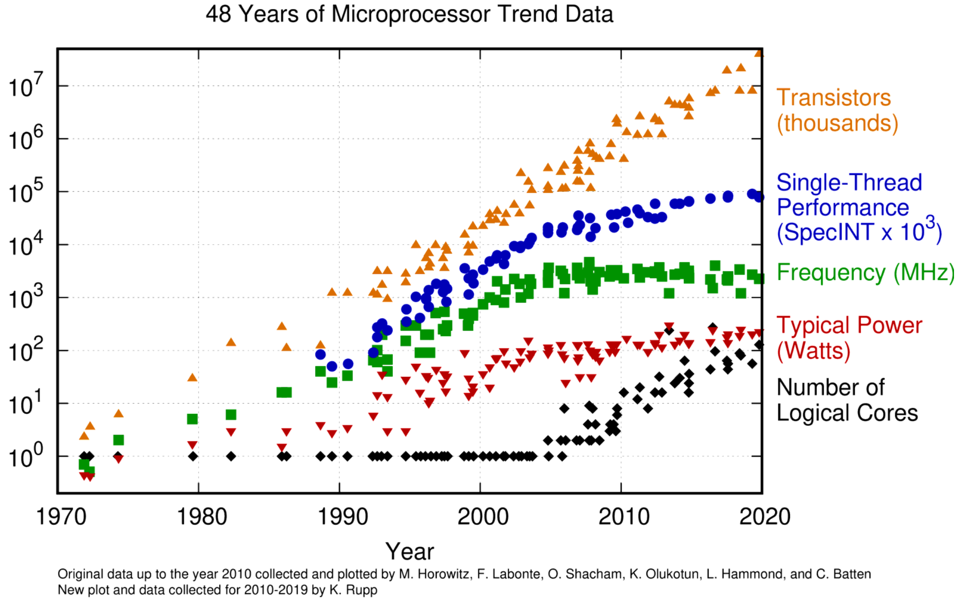


Figure 2.2 Microprocessors follow Moore's law by increasing the number of cores, while single thread performance and total power are saturating.

The Illumina DRAGEN Bio-IT [24] Platform which is developed on FPGAs offers more efficient genomics sequencing analysis. The Falcon Accelerated Genomics Pipeline (FAGP) [25] also leverages the power of FPGAs to accelerate genome processing. Genetalks [26] provides FPGA based variant calling solutions for GATK best practices pipelines. NVIDIA Clara Parabricks [27] variant calling pipelines which are accelerated on GPUs also provide improved performance as compared to CPU only processing.

2.4.3. Applications-specific integrated circuit (ASIC)

FPGA designs are considered flexibile, which means their hardware design is easy to change or adapt with new updates in the algorithms. FPGA designs are generally considered good for prototyping purposes. However, ASICs are the ultimate product level solutions for higher performance, lower power consumption and cost efficiency (when produced in bulk). ASIC applications remain mainly limited to embedded systems, but new data-driven and computer-vision technologies are pushing towards a faster development and replacement of general-purpose computing with ASICs. Google's Tensor Processing Unit (TPU) [28] and YouTube's Video (trans)Coding Unit (VCU) [29] are the latest examples of ASICs used for machine learning and video processing applications, respectively. The latest Green500 list top, the MN-3 HPC supercomputer, uses their custom ASIC, MN-Core [30] (Accelerator for Deep Learning). Genomics is still a young field, where new algorithmic approaches are being developed regularly. Therefore,

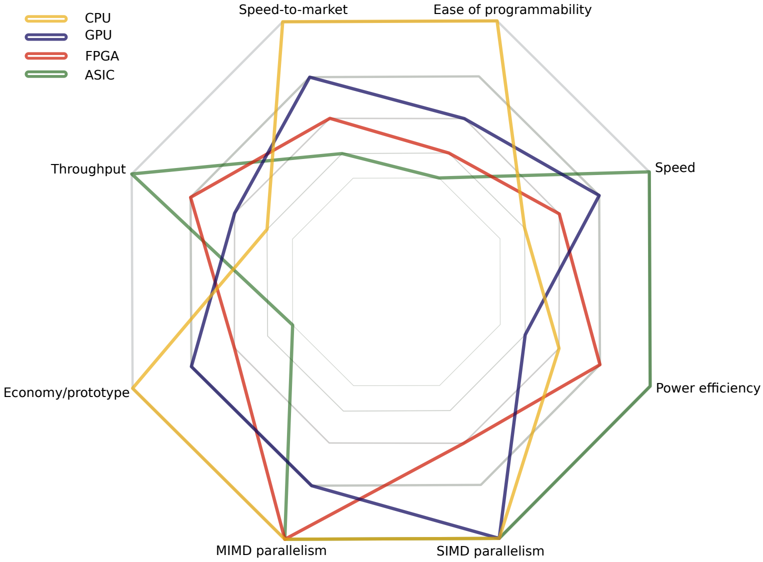


Figure 2.3 A comprehensive comparison of CPUs, GPUs, FPGAs and ASICs from a performance, parallelization, cost and throughput perspective.

it is too early to say when ASICs will make their way into genome processing. However, ASICs represent the ultimate solution to enable high-performance and low-power processing. In addition, the application specific nature of genomics pipelines, where applications are focused on identifying the variations in a genome, makes them particularly well-suited for ASIC processing rather than general purpose processing. Here, we mention some early efforts in ASIC development for genome assembly and alignment applications. Darwin [31] performs reference-guided and de-novo assembly, created in TSMC 40nm CMOS process. Total calculated power consumption for this ASIC device is 15W while maintaining an area of $412\,\mathrm{mm}^2$. In modern technology nodes like 14nm, this design will consume 3x less power and 6x less area as reported in their work. A more detailed and comprehensive comparison of CPUs, GPUs, FPGAs and ASICs for performance capability, achievable parallelization, throughput, latency and power consumption, product total cost and development time is given in Figure 2.3.

2.5. Genome analysis on distributed and parallel computing systems

Generally genomics data formats permit independent compute and analytic operations on a granular level, i.e., even smaller chunks can be processed without any dependency issues with other cascaded data. This eventually helps to run genome analysis algorithms on multiple data chunks in parallel. So storing genomics datasets in distributed file systems and processing it in a parallel and distributed compute environment offers both flexibility and efficiency. This section outlines some popular distributed and parallel computing systems architectures, their programming models, data management and analytic platforms. Development and adoption of methods for scalability of genomics applications on such systems are also discussed.

2.5.1. High performance computing (HPC) and supercomputers

HPC infrastructure combines compute resources and storage by connecting them through a highly parallel and optimized network to exploit parallel programming models for high throughput and efficient large scale computing applications in a non-distributed environment. Mainly HPC systems are deployed by R&D centers and governmental organizations with large capital investment and maintenance costs. The TOP500 list measures the theoretical peak performance of the world top 500 supercomputers bi-annually reported in floating-point operations per second (FLOPS) performance on the LINPACK Benchmark.

2.5.2. Parallel programming models (MPI, SHMEM & PGAS)

A lot of parallel programming models for HPC applications have been developed. OpenMP is the most common shared-memory parallel programming model for intra-node processing where tasks share a common address space, while the message passing interface (MPI) is used in distributed memory systems for inter-node processing where tasks have private memories and they communicate with each other through message exchange. A hybrid of both models MPI-X (MPI+OpenMP) is often preferred for performance efficiency on both inter-node and intra-node applications. MPI-RMA (remote memory access) and SHMEM (shared-memory) programming models allow one-sided communication of data without process synchronization. Partitioned global address space (PGAS) is a shared and distributed memory programming model where a portion of logically partitioned address space is shared between distributed processes or threads. Figure 2.4 shows an overview of the architectural differences in different parallel programming models in the shared and distributed memory context for processes. There are many cluster scaled implementations available for alignment using both MPI and PGAS parallel programming models. pBWA [32] and mpi-

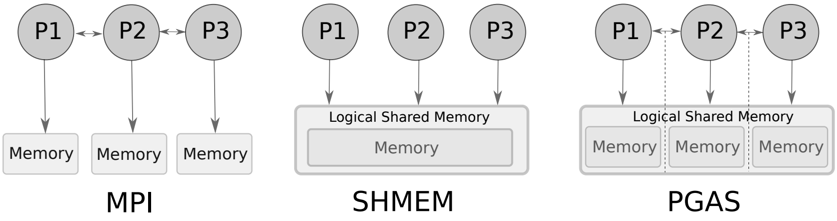


Figure 2.4 Popular parallel programming models for shared and distributed memory interfaces for HPC: MPI (MPI-X, MPI-RMA), SHMEM, PGAS.

BLAST [33] use MPI while CUSHAW3 [34] is based on UPC++ (a language based on PGAS). One of the latest algorithms in this domain is QUARTIC mpiBWA [35], which is a distributed BWA-MEM alignment algorithm by employing MPI functionality and uses MPI shared I/O for input/output on a parallel file system.

2.5.3. Data centers and cloud computing

As organizations strive to provide fast and uninterrupted internet communication, compute and storage infrastructure to information technology (IT) operations and telecommunications services, maintaining such infrastructure under a single data center entity can be an efficient and cost effective way for emerging compute demands. Data centers provide high availability, modularity and flexibility to their customers. Data centers also host cloud computing components, which enables on-demand availability and delivery of flexible compute resources (processing, software, storage, network and databases) through the internet. Public clouds services are offered by third-party cloud providers (Google Cloud, Amazon AWS, Alibaba Cloud, Huawei Cloud, Microsoft Azure) to anyone on the public internet. These vendors allow customers to pay per usage of subscribed services. On the other hand, private clouds offer compute resources through the internet or private internal networks to specific customers or users. Private clouds give an additional benefit of customization and control of on-premise hosted computing resources with more security and privacy assurances. These clouds can deliver services such as infrastructure as a service (IaaS) or platform as a service (PaaS) offerings. Some data center vendors can even provide HPC-as-a-service solutions in the cloud.

2.5.4. Big data analytics frameworks (Hadoop, Spark, Arrow)

The distributed data-parallel programming model MapReduce has been widely used in big data frameworks where parallel data processing is achieved by managing communications and data shuffling inside systems through redundancy and fault tolerance. Apache Hadoop is an open-source implementation of such programming paradigm which uses Hadoop distributed file system (HDFS) to manage big data processing on a parallel and distributed system. Apache Spark is a unified analytics engine to process streaming and batch big data processing in a distributed computing environment, with built-in modules for streaming data, distributed machine learning, SQL functions and graph processing. Spark also provides a high-level APIs for the Java, Scala, Python and R languages. In Spark, resilient distributed datasets (RDDs) are the core components that are distributed across the nodes of a cluster to be operated on in parallel. RDDs can be cached/persisted in-memory across nodes to store intermediate results for iterative processing. Spark commonly uses HDFS to read/write data, but also supports other storage systems like NFS, HBase and Amazon's S3. ADAM's Cannoli [36] and SparkBWA [37] are Apache Spark based BWA-MEM implementations that use BWA-MEM as a loosely integrated application underneath these implementations, while GATK BWASpark [38] modifies the original BWA-MEM to exploit the Spark

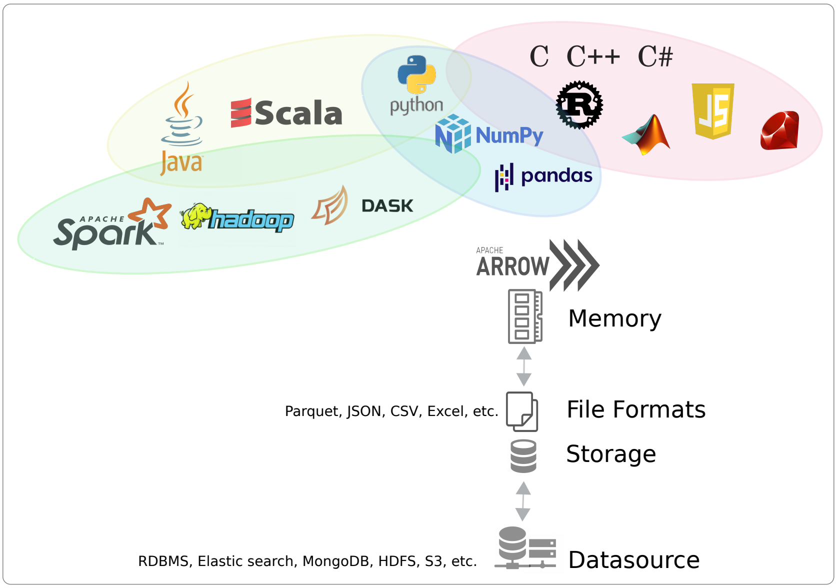


Figure 2.5 The Apache Arrow eco-system provides a de-facto standard columnar and in-memory data format for data analytics on a range of languages and big data frameworks like Apache Spark, Hadoop and DASK.

scheduling and shuffling operations to run BWA-MEM instances in parallel on clusters. On the other hand, the GATK pipeline is scaled up by Halvade [39], which uses the Hadoop MapReduce API, while ADAM [36] and SparkGA2 [40] use the Apache Spark framework and HDFS as a distributed file system. These are a few of the implementations that can handle whole variant calling workflows based on GATK best practices including alignment, sorting, duplicates removal, base quality score recalibration and variant callers. Apache Arrow is an in-memory standard columnar data format which provides API interfaces and functions to process analytics workloads in the Go, C, C++, C#, Java, JavaScript, R, Rust, MATLAB, Ruby and Python languages. Due to the columnar data storage format, efficient vectorized data analytics operations and better cache locality can be exploited. This in-memory format also supports zero-copy reads for large datasets in inter-process communication without serialization/deserialization overheads. VC@Scale [41], a DeepVariant based variant calling workflow also uses Apache Spark in-conjunction with Apache Arrow as an additional layer for columnar data format and columnar vectorized processing in PySpark. Figure 3.5 elaborates on how different programming languages and big data frameworks make use of a single standard and unified columnar data format with a complete set of Arrow enabled APIs and compute kernels in the eco-system.

2.6. Data, compute and sequencing: A perspective

In the past decades particularly, advances in semiconductor technologies enabled explosive (exponential) increase in data generation. Broadband cellular telecommunication networks (4G, 5G), industrial and home automation technologies like cyber-physical systems (CPS) and Internet of Things (IoT) devices, cloud computing and storage infrastructures – artificial intelligence (AI) and machine learning (ML) systems are being trained on more diverse and big amount of data for improved prediction accuracy and classification on population scale analysis. Data is now an essential part of new online business models, particularly on social media platforms to monitor and analyze real-time social and behavioral tendencies of customers to present related advertisement. To anticipate customer behavior and market trends for business development and strategies, financial institutions heavily rely on their own or third-party provided datasets. Image datasets have gained a lot of value in this digital society. Satellite imagery is used in defense, mapping and even for real-time observations. Time-lapsed multi-satellites and/or collaborative imagery datasets can also be used to estimate/simulate the past/future impact of climate change or to study other related environmental impact. Medical imaging in the form of X-rays, CT-scans, Ultrasound and MRI is being used to detect complex diseases using AI & ML methods for more accurate and precise diagnostic. Similarly existing datasets and trained models or models to be trained with one's own dataset in transportation, power grids and cities planning and modeling have become an increasingly important part of this new data-oriented or more precisely data-driven digital society. Genomics data follows the same growth trends as other big dataset, and is estimated to produce between 2 and 40 exabytes of data in the next decade through research activities only. As reported in [42], sequencing data being produced worldwide is doubling every 7 months. Once sequencing becomes normal practice in clinical settings, genomics data will require sufficient resources to process this data in a reasonable amount of time. As shown in Figure 2.6, raw sequencing data alone deposited in the three main sequencing archives GenBank, WGS and SRA exceeds 6.12E-05 zetta Bases. At the same time, sequencing costs are falling rapidly as shown in Figure 2.7, just in two decades the costs fell from $95 million to around $500 for a single human genome. This makes sequencing more accessible as well as requires smart computing infrastructure for cost effective and efficient downstream analysis. Although cloud computing is an immediate solution for processing large amounts of data, there are more sustainable alternatives represented by embedded sequencing, data processing and analysis.

Data growth is outpacing computing growth. Big data growth is estimated to double every two years, thereby creating every two years an amount of data that is comparable to the amount of data that has been created in the whole of recorded history. Figure 2.7 shows that the data volume created, captured, consumed and/or copied worldwide is estimated more than 58 zetta Bytes in the 2021 alone. At the same time, with the limitations to maintain the pace of Dennard's scaling (which refers to reducing transistor size that decreases total chip area while keeping a constant power density) and at the twilight of Moore's

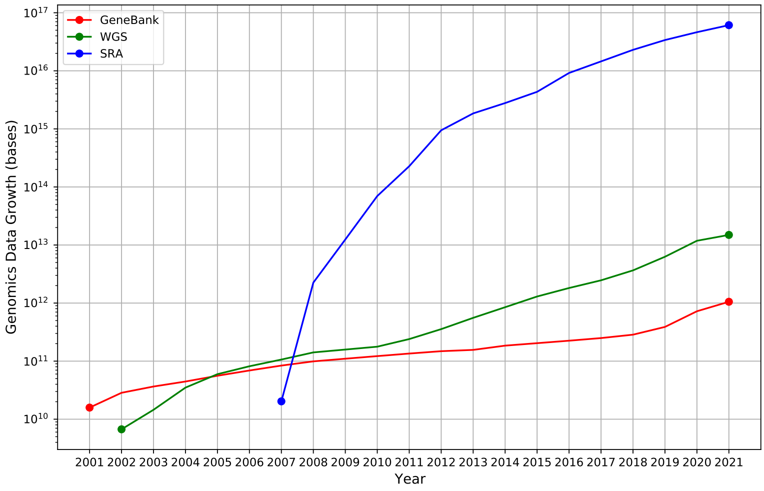


Figure 2.6 A log scale plot of sequencing data deposited on GenBank (NIH genetic sequence database), Whole Genome Shotgun (WGS) and Sequence Read Archive (SRA) repositories.

law (which refers to the doubling of the number of transistors on a chip every two years), this explosive growth of data poses a serious challenge to compute performance. To maintain an adequate and sustainable compute performance growth for future needs we have to think differently.

Cloud computing is being considered an inevitable solution for big data processing. Users and businesses can upload raw data from edge devices to the cloud and can process huge datasets in real-time. The worldwide public cloud computing market has grown from 6 to 250 billion US dollars in just the last decade. However, this rapidly growing field also poses challenges to the environment in-term of energy consumption. The latest top edition to the Top500 list, the Fugaku supercomputer, which is powered by Fujitsu's 48-core A64FX SoC operating at 2.2GHz frequency gives an energy performance of 14.78 GFLOPS/watt while the top contender on the Green500 list, the MN-3 from Preferred Networks (PFN) powered by Intel Xeon Platinum 8260M, operates at 2.4GHz and is able to achieve 39.379 GFLOPS/watt. This represents about a 3x difference in the power efficiency of these two supercomputing systems. Figure 2.7 shows a logarithmic plot of the sum of overall top500 worldwide supercomputers performance development over the last two decades. The projected performance development line shows that in last couple of years this become difficult to maintain the processing performance improvements at the same pace. Both in the cloud as well as for HPC infrastructures, design for energy efficiency should be considered as a priority. This is necessary for providing energy efficiency to combat escalating

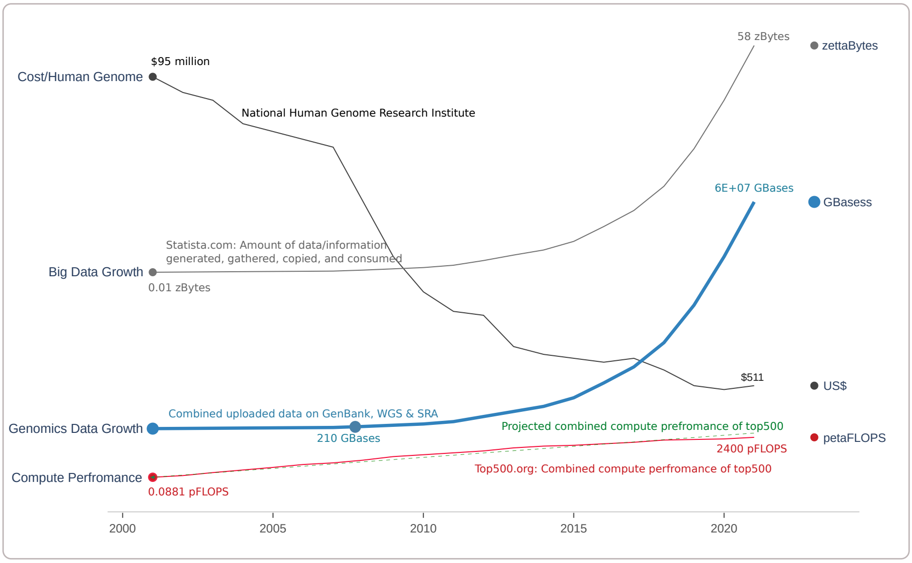


Figure 2.7 Plot of human genome sequencing costs, big data and genomics data growth, and compute performance (log scale) growth over the last two decades.

global energy consumption and ensure environmental protection.

RISC-V, an open-source processor ISA, provides an excellent opportunity for processors in the embedded system design market to exploit and unearth potential innovations in both research and industry. Developers in open-source communities are more open to share and develop new ideas. This open revolution can lead to many new processor architecture and design innovations even beyond the von Neumann architecture to meet the future compute needs more efficiently.

FPGA and ASIC-based solutions for genome analysis exhibit enormous advantages over traditional CPU based analysis both in-terms of performance and power efficiency. The Illumina DRAGEN Bio-IT Platform, which is developed on FPGA, offers more efficient genomics sequencing analysis on-premise as well as in the cloud. Falcon Accelerated Genomics Pipeline (FAGP) also leverage the power of FPGAs to accelerate the genome processing on Intel Programmable Acceleration Card (PAC) on Dell EMC machines. Genetalks also provides FPGA based variant calling solutions for GATK best practices pipelines using the Xilinx Alveo Acceleration Card in the cloud. These solutions give a promising prospect on how FPGA/ASIC can revolutionize the computing challenges in genome sequencing analysis.

REFERENCES

[1] H. Li and R. Durbin. "Fast and accurate short read alignment with Burrows–Wheeler transform". In: *Bioinformatics* 25.14 (May 2009), pp. 1754–1760. issn: 1367-4803. doi: 10.1093/bioinformatics/btp324. url: https://doi.org/10.1093/bioinformatics/btp324.

[2] B. Langmead and S. L. Salzberg. "Fast gapped-read alignment with Bowtie 2". In: *Nature Methods* 9.4 (Apr. 2012), pp. 357–359. issn: 1548-7105. doi: 10.1038/nmeth.1923. url: https://doi.org/10.1038/nmeth.1923.

[3] H. Li. "Minimap and miniasm: fast mapping and de novo assembly for noisy long sequences". In: *Bioinformatics* 32.14 (July 2016). 27153593[pmid], pp. 2103–2110. issn: 1367-4811. doi: 10.1093/bioinformatics/btw152. url: https://www.ncbi.nlm.nih.gov/pubmed/27153593.

[4] C. Jain, A. Rhie, H. Zhang, C. Chu, B. P. Walenz, S. Koren, and A. M. Phillippy. "Weighted minimizer sampling improves long read mapping". In: *Bioinformatics* (July 2020). issn: 1367-4803. doi: 10.1093/bioinformatics/btaa435. url: https://doi.org/10.1093/bioinformatics/btaa435.

[5] J. Ren and M. J. P. Chaisson. "lra: A long read aligner for sequences and contigs". In: *PLOS Computational Biology* 17.6 (June 2021), pp. 1–23. doi: 10.1371/journal.pcbi.1009078. url: https://doi.org/10.1371/journal.pcbi.1009078.

[6] K. Shafin. "Nanopore sequencing and the Shasta toolkit enable efficient de novo assembly of eleven human genomes". In: *Nature Biotechnology* 38.9 (Sept. 2020), pp. 1044–1053. issn: 1546-1696. doi: 10.1038/s41587-020-0503-6. url: https://doi.org/10.1038/s41587-020-0503-6.

[7] J. Ruan and H. Li. "Fast and accurate long-read assembly with wtdbg2". In: *Nature Methods* 17.2 (Feb. 2020), pp. 155–158. issn: 1548-7105. doi: 10.1038/s41592-019-0669-3. url: https://doi.org/10.1038/s41592-019-0669-3.

[8] M. Kolmogorov, J. Yuan, Y. Lin, and P. A. Pevzner. "Assembly of long, error-prone reads using repeat graphs". In: *Nature Biotechnology* 37.5 (May 2019), pp. 540–546. issn: 1546-1696. doi: 10.1038/s41587-019-0072-8. url: https://doi.org/10.1038/s41587-019-0072-8.

[9] H. Cheng, G. T. Concepcion, X. Feng, H. Zhang, and H. Li. “Haplotype-resolved de novo assembly using phased assembly graphs with hifiasm”. In: *Nature Methods* 18.2 (Feb. 2021), pp. 170–175. issn: 1548-7105. doi: 10.1038/s41592-020-01056-5. url: https://doi.org/10.1038/s41592-020-01056-5.

[10] S. Koren, B. P. Walenz, K. Berlin, J. R. Miller, N. H. Bergman, and A. M. Phillippy. “Canu: scalable and accurate long-read assembly via adaptive k-mer weighting and repeat separation”. In: *Genome Research* 27.5 (Mar. 2017), pp. 722–736. doi: 10.1101/gr.215087.116. url: https://doi.org/10.1101/gr.215087.116.

[11] B. Institute. *GATK Variant Calling Pipelines*. 2010. url: https://software.broadinstitute.org/gatk/best-practices/.

[12] E. Garrison and G. Marth. *Haplotype-based variant detection from short-read sequencing*. 2012. eprint: arXiv:1207.3907, [Accessed:11April2019].

[13] D. C. Koboldt, Q. Zhang, D. E. Larson, D. Shen, M. D. McLellan, L. Lin, C. A. Miller, E. R. Mardis, L. Ding, and R. K. Wilson. “VarScan 2: Somatic mutation and copy number alteration discovery in cancer by exome sequencing”. In: *Genome Research* 22.3 (Feb. 2012), pp. 568–576. doi: 10.1101/gr.129684.111. url: https://doi.org/10.1101/gr.129684.111.

[14] Z. Lai, A. Markovets, M. Ahdesmaki, B. Chapman, O. Hofmann, R. McEwen, J. Johnson, B. Dougherty, J. C. Barrett, and J. R. Dry. “VarDict: a novel and versatile variant caller for next-generation sequencing in cancer research”. In: *Nucleic Acids Research* 44.11 (Apr. 2016), e108–e108. doi: 10.1093/nar/gkw227. url: https://doi.org/10.1093/nar/gkw227.

[15] K. Cibulskis, M. S. Lawrence, S. L. Carter, A. Sivachenko, D. Jaffe, C. Sougnez, S. Gabriel, M. Meyerson, E. S. Lander, and G. Getz. “Sensitive detection of somatic point mutations in impure and heterogeneous cancer samples”. In: *Nature Biotechnology* 31 (Feb. 2013), 213 EP -. url: https://doi.org/10.1038/nbt.2514.

[16] Z. Wei, W. Wang, P. Hu, G. J. Lyon, and H. Hakonarson. “SNVer: a statistical tool for variant calling in analysis of pooled or individual next-generation sequencing data”. In: *Nucleic Acids Research* 39.19 (Aug. 2011), e132–e132. doi: 10.1093/nar/gkr599. url: https://doi.org/10.1093/nar/gkr599.

[17] A. Wilm, P. P. K. Aw, D. Bertrand, G. H. T. Yeo, S. H. Ong, C. H. Wong, C. C. Khor, R. Petric, M. L. Hibberd, and N. Nagarajan. “LoFreq: a sequence-quality aware, ultra-sensitive variant caller for uncovering cell-population heterogeneity from high-throughput sequencing datasets”. In: *Nucleic Acids Research* 40.22 (Oct. 2012), pp. 11189–11201. doi: 10.1093/nar/gks918. url: https://doi.org/10.1093/nar/gks918.

[18] T. Dunn, G. Berry, D. Emig-Agius, Y. Jiang, S. Lei, A. Iyer, N. Udar, H.-Y. Chuang, J. Hegarty, M. Dickover, B. Klotzle, J. Robbins, M. Bibikova, M. Peeters, and M. Strömberg. "Pisces: an accurate and versatile variant caller for somatic and germline next-generation sequencing data". In: *Bioinformatics* 35.9 (Oct. 2018). Ed. by R. Schwartz, pp. 1579–1581. doi: 10.1093/bioinformatics/bty849. url: https://doi.org/10.1093/bioinformatics/bty849.

[19] S. Kim, K. Scheffler, A. L. Halpern, M. A. Bekritsky, E. Noh, M. Källberg, X. Chen, Y. Kim, D. Beyter, P. Krusche, and C. T. Saunders. "Strelka2: fast and accurate calling of germline and somatic variants". In: *Nature Methods* 15.8 (2018), pp. 591–594. issn: 1548-7105. doi: 10.1038/s41592-018-0051-x. url: https://doi.org/10.1038/s41592-018-0051-x.

[20] R. Poplin, P.-C. Chang, D. Alexander, S. Schwartz, T. Colthurst, A. Ku, D. Newburger, J. Dijamco, N. Nguyen, P. T. Afshar, S. S. Gross, L. Dorfman, C. Y. McLean, and M. A. DePristo. "A universal SNP and small-indel variant caller using deep neural networks". In: *Nature Biotechnology* 36 (Sept. 2018). url: https://doi.org/10.1038/nbt.4235.

[21] M. U. Ahsan, Q. Liu, L. Fang, and K. Wang. "NanoCaller for accurate detection of SNPs and indels in difficult-to-map regions from long-read sequencing by haplotype-aware deep neural networks". In: *Genome Biology* 22.1 (Sept. 2021), p. 261. issn: 1474-760X. doi: 10.1186/s13059-021-02472-2. url: https://doi.org/10.1186/s13059-021-02472-2.

[22] S. M. E. Sahraeian, R. Liu, B. Lau, K. Podesta, M. Mohiyuddin, and H. Y. K. Lam. "Deep convolutional neural networks for accurate somatic mutation detection". In: *Nature Communications* 10.1 (Mar. 2019), p. 1041. issn: 2041-1723. doi: 10.1038/s41467-019-09027-x. url: https://doi.org/10.1038/s41467-019-09027-x.

[23] M. Vasimuddin, S. Misra, H. Li, and S. Aluru. "Efficient Architecture-Aware Acceleration of BWA-MEM for Multicore Systems". In: *2019 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. 2019, pp. 314–324. doi: 10.1109/IPDPS.2019.00041.

[24] A. Goyal, H. J. Kwon, K. Lee, R. Garg, S. Y. Yun, Y. H. Kim, S. Lee, and M. S. Lee. "Ultra-Fast Next Generation Human Genome Sequencing Data Processing Using DRAGENTM Bio-IT Processor for Precision Medicine". In: *Open Journal of Genetics* 07.01 (2017), pp. 9–19. doi: 10.4236/ojgen.2017.71002. url: https://doi.org/10.4236/ojgen.2017.71002.

[25] DELL. *Boost Genomic Sequencing with Falcon Accelerated Genomics Pipeline (FAGP) on Intel FPGA PAC*. 2021. url: https://www.dell.com/support/kbdoc/nl-nl/000136278/boost-genomic-sequencing-with-falcon-accelerated-genomics-pipeline-fagp-on-intel-fpga-pac.

[26] GeneTalks. *GTX.FPGA - Bioinformatics Acceleration, High Performance Genomic Computing*. 2021. url: https://www.xilinx.com/publications/solution-briefs/partner/genetalk-solution-brief.pdf.

[27] Nvidia. *NVIDIA Clara Parabricks: A GPU-accelerated computational genomics application framework*. 2021. url: https://www.nvidia.com/en-us/clara/genomics/.

[28] Google. *Tensor Processing Units (TPUs)*. 2022. url: https://cloud.google.com/tpu/docs/tpus.

[29] P. Ranganathan, D. Stodolsky, J. Calow, J. Dorfman, M. Guevara, C. W. Smullen IV, A. Kuusela, R. Balasubramanian, S. Bhatia, P. Chauhan, A. Cheung, I. S. Chong, N. Dasharathi, J. Feng, B. Fosco, S. Foss, B. Gelb, S. J. Gwin, Y. Hase, D.-k. He, C. R. Ho, R. W. Huffman Jr., E. Indupalli, I. Jayaram, P. Kongetira, C. M. Kyaw, A. Laursen, Y. Li, F. Lou, K. A. Lucke, J. Maaninen, R. Macias, M. Mahony, D. A. Munday, S. Muroor, N. Penukonda, E. Perkins-Argueta, D. Persaud, A. Ramirez, V.-M. Rautio, Y. Ripley, A. Salek, S. Sekar, S. N. Sokolov, R. Springer, D. Stark, M. Tan, M. S. Wachsler, A. C. Walton, D. A. Wickeraad, A. Wijaya, and H. K. Wu. "Warehouse-Scale Video Acceleration: Co-Design and Deployment in the Wild". In: *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*. ASPLOS 2021. Virtual, USA: Association for Computing Machinery, 2021, pp. 600–615. isbn: 9781450383172. doi: 10.1145/3445814.3446723. url: https://doi.org/10.1145/3445814.3446723.

[30] P. N. (PFN). *MN-Core (Accelerator for Deep Learning)*. 2022. url: https://projects.preferred.jp/mn-core/#mn-core.

[31] Y. Turakhia, G. Bejerano, and W. J. Dally. "Darwin: A Genomics Co-Processor Provides up to 15,000X Acceleration on Long Read Assembly". In: *SIGPLAN Not.* 53.2 (Mar. 2018), pp. 199–213. issn: 0362-1340. doi: 10.1145/3296957.3173193. url: https://doi.org/10.1145/3296957.3173193.

[32] L. X, K. Qiu, P. Liang, and P. D. "Speeding up large-scale next generation sequencing data analysis with pBWA". In: *Journal of biocomputing* 1 (Jan. 2012). doi: 10.4172/jbcg.1000101.

[33] A. Darling, L. Carey, and W. Feng. "The Design, Implementation, and Evaluation of mpiBLAST". In: *Proc Cluster World* 2003 (Dec. 2003).

[34] Y. Liu, B. Popp, and B. Schmidt. "CUSHAW3: Sensitive and Accurate Base-Space and Color-Space Short-Read Alignment with Hybrid Seeding". In: *PLOS ONE* 9.1 (Jan. 2014), pp. 1–9. doi: 10.1371/journal.pone.0086869. url: https://doi.org/10.1371/journal.pone.0086869.

[35] Frédéric, N. Joly, N. Fedy, T. Magalhaes, L. Sirotti, P. Paganiban, F. Martin, M. McManus, and P. Hupé. “QUARTIC: QUick pArallel algoRithms for high-Throughput sequencIng data proCessing”. In: *F1000Research* 9 (Oct. 2020), p. 240. doi: 10.12688/f1000research.22954.3.

[36] M. Massie, F. Nothaft, C. Hartl, C. Kozanitis, A. Schumacher, A. D. Joseph, and D. A. Patterson. *ADAM: Genomics Formats and Processing Patterns for Cloud Scale Computing*. Tech. rep. UCB/EECS-2013-207, EECS Department, University of California, Berkeley, 2013.

[37] B. Institute. *BWA on Spark*. 2018. url: https://gatk.broadinstitute.org/hc/en-us/articles/360037225092-BwaSpark-BETA-.

[38] B. Institute. *GATK4 on Spark*. 2018. url: https://software.broadinstitute.org/gatk/documentation/article?id=11245.

[39] D. Decap, J. Reumers, C. Herzeel, P. Costanza, and J. Fostier. “Halvade: scalable sequence analysis with MapReduce”. eng. In: *Bioinformatics (Oxford, England)* 31.15 (Aug. 2015). btv179[PII], pp. 2482–2488. issn: 1367-4811. doi: 10.1093/bioinformatics/btv179. url: https://doi.org/10.1093/bioinformatics/btv179.

[40] H. Mushtaq, F. Liu, C. Costa, G. Liu, P. Hofstee, and Z. Al-Ars. “SparkGA: A Spark Framework for Cost Effective, Fast and Accurate DNA Analysis at Scale”. In: *Proceedings of the 8th ACM International Conference on Bioinformatics, Computational Biology,and Health Informatics*. ACM-BCB ’17. Boston, Massachusetts, USA: ACM, 2017, pp. 148–157. isbn: 978-1-4503-4722-8. doi: 10.1145/3107411.3107438. url: http://doi.acm.org/10.1145/3107411.3107438.

[41] T. Ahmad, Z. Al Ars, and H. P. Hofstee. “VC@Scale: Scalable and high-performance variant calling on cluster environments”. In: *GigaScience* 10.9 (Sept. 2021). giab057. issn: 2047-217X. doi: 10.1093/gigascience/giab057. eprint: https://academic.oup.com/gigascience/article-pdf/10/9/giab057/40327053/giab057.pdf. url: https://doi.org/10.1093/gigascience/giab057.

[42] Z. D. Stephens, S. Y. Lee, F. Faghri, R. H. Campbell, C. Zhai, M. J. Efron, R. Iyer, M. C. Schatz, S. Sinha, and G. E. Robinson. “Big Data: Astronomical or Genomical?” In: *PLOS Biology* 13.7 (July 2015), pp. 1–11. doi: 10.1371/journal.pbio.1002195. url: https://doi.org/10.1371/journal.pbio.1002195.

3

Node-level performance optimizations using Apache Arrow

Prelude: *This chapter introduces node level performance optimization techniques for germline variant calling pipelines. It includes a short overview of Apache Arrow, a columnar in-memory data format, ArrowSAM (Apache Arrow based format for in-memory SAM data storage and computation), GATK best practices variant calling pipeline and issues related to its performance scalability. More specifically, Section 3.1 discusses ArrowSAM integration into variant calling pre-processing applications including alignment, sorting and duplicate removal, while Section 3.2 describes ArrowSAM integration into the whole GATK best practices pipeline. Performance optimizations in terms of both runtime speedup as well as better system resource utilization are also discussed. The contents of this chapter are based on our conference paper entitled "ArrowSAM: In-Memory Genomics Data Processing Using Apache Arrow [1]" published in IEEE, ICCAIS in 2020 and the journal article entitled "Optimizing performance of GATK workflows using Apache Arrow In-Memory data framework [2]" which was published in BMC Genomics in 2020.*

3.1. ArrowSAM and variant calling pre-processing

The rapidly growing size of genomics datasets, driven by advances in sequencing technologies, demands fast and cost-effective processing. However, processing this data creates many challenges, particularly in selecting appropriate algorithms and computing platforms. Computing systems need data closer to the processor for fast processing. Traditionally, due to cost, volatility and other physical constraints of DRAM, it was not feasible to place large amounts of working data sets in memory. However, new emerging storage class memories allow storing and processing big data closer to the processor. In this work, we show how the commonly used genomics data format, Sequence Alignment/Map (SAM), can be presented in the Apache Arrow in-memory data representation to benefit of in-memory processing and to ensure better scalability through shared memory objects, by avoiding large (de)-serialization overheads in cross-language interoperability. To demonstrate the benefits of such a system, we propose ArrowSAM, an in-memory SAM format that uses the Apache Arrow framework, and integrate it into genome pre-processing pipelines including BWA-MEM, Picard and Sambamba. Results show 15x and 2.4x speedups as compared to Picard and Sambamba, respectively. The code and scripts for running all workflows are freely available at: https://github.com/abs-tudelft/ArrowSAM

3.1.1. Introduction

Genomics is projected to generate the largest big data sets globally, which requires modifying existing tools to take advantage of new developments in big data analytics and new memory technologies to ensure better performance and high throughput. In addition, new applications using DNA data are becoming ever more complex, such as the study of large sets of complex genomics events like gene isoform reconstruction and sequencing large numbers of individuals with the aim of fully characterizing genomes at high resolution [3]. This underscores the need for efficient and cost effective DNA analysis infrastructures.

At the same time, genomics is a young field. To process and analyze genomics data, the research community is actively working to develop new, efficient and optimized algorithms, techniques and tools, usually programmed in a variety of languages, such as C, Java or Python. These tools share common characteristics that impose limitations on the performance achievable by the genomics pipelines.

- These tools are developed to use traditional I/O file systems which incur a huge I/O bottleneck in computation due to disk bandwidth [4]. Each tool reads from the I/O disks, computes and writes back to disk.

- Due to the virtualized nature of some popular languages used to develop genomics tools (such as Java), these tools are not well suited to exploit modern hardware features like Single-instruction multiple-data (SIMD) vectorization and accelerators (GPU or FPGAs).

This paper proposes a new approach for representing genomics data sets, based on recent developments in big data analytics to improve the performance and efficiency of genomics pipelines. Our approach consists of the following main contributions:

- We propose an in-memory SAM data representation, called ArrowSAM, created in Apache Arrow to place genome data in RecordBatches of immutable shared memory objects for inter-process communication. We use DRAM for ArrowSAM placement and inter-process access.
- We adapt existing widely-used genomics data pre-processing applications (for alignment, sorting and duplicates removal) to use the Apache Arrow framework and to benefit from immutable shared memory plasma objects in inter process communication.
- We compare various workflows for genome pre-processing, using different techniques for in-memory data communication and placement (for intermediate applications), and show that ArrowSAM in-memory columnar data representation outperforms.

The rest of this paper is organized as follows. Section 3.1.2 discusses background information on genomics tools and Apache Arrow big data format as well as presents the new ArrowSAM genomics data format. Section 3.1.3 shows how to integrate Apache Arrow into existing genomics tools, while Section 3.1.4 discusses the measurement results of these new tools. Section 3.1.5 elaborates CPU usage, memory access and usage patterns. Section 3.1.6 presents related work in the field. Finally, Section 3.1.7 ends with the conclusions.

3.1.2. Background

This section provides a short description of DNA sequence data pre-processing tools, followed by a brief introduction to the Apache Arrow framework.

DNA pre-processing: After DNA data is read by sequencing machines, alignment tools align reads to the different chromosomes of a reference genome and generate an output file in the SAM format. BWA-MEM [5] is a widely used tools for this purpose. The generated SAM file describes various aspects of the alignment result, such as map position and map quality. SAM is the most commonly used alignment/mapping format. To eliminate some systematic errors in the reads, some additional data pre-processing and cleaning steps are subsequently performed, like sorting the reads according to their chromosome name and position. Picard [6] and Sambamba [7] are some tools commonly used for such operations. This is followed by the mark duplicates step, where duplicate reads are removed by comparing the reads having the same map positions and orientation and selecting the read with the highest quality score. Duplicate reads are generated due to the wetlab procedure of creating multiple copies of DNA molecules to make sure there are enough samples of each molecule to facilitate the sequencing process. Again, Picard and Sambamba are commonly used here.

Apache Arrow: To manage and process large data sets, many different big data frameworks have been created. Some examples include Apache Hadoop, Spark, Flink, MapReduce and Dask. These frameworks provide highly scalable, portable and programmable environments to improve storage and scalability of big data analytics pipelines. They are generally built on top of high-level language frameworks such as Java and Python to ensure ease of programmability. However, such high-level languages induce large processing overheads, forcing programmers to resort to low-level languages such as C to process specific computationally intensive parts of big data applications. On the other hand, heterogeneous components like FPGAs and GPUs are being increasingly used in cluster and cloud computing environments to improve performance of big data processing. These components are, however, programmed using very close-to-metal languages like C/C++ or even hardware-description languages. The multitude of technologies used often results in a highly heterogeneous system stack that can be hard to optimize for performance. However, combining processes programmed in different languages induces large inter-process communication overheads (so called data (de)serialization) whenever the data is moved between such processes.

To mitigate this problem, the Apache Arrow [8] project was initiated to provide an open standardized format and interfaces for tabular data in-memory. Using language-specific libraries, multiple languages can share in-memory data without any copying or serialization. This is done using the plasma inter-process communication component of Arrow, that handles shared memory pools across different processes in a pipeline [9].

ArrowSAM data format: This paper proposes a new in-memory genomics SAM format based on Apache Arrow. Such a representation can benefit from two aspects to improve overall system throughout: one is related to the tabular nature of genomics data and the other related to cross-language interoperability. Using Arrow ensures efficient genomics data placement in memory to gain maximum throughput and parallel data access efficiency. The tabular genomics data format can benefit from the standardized, cross-languages in-memory data representation of Arrow, that provides insight into the organization of the data sets through its schema.

In order to enable genomics applications to use Apache Arrow, two different contributions are needed. First, we need to define an Arrow in-memory representation of the corresponding genomics SAM data format. Second, the applications and tools using the data need to be adapted to access the new format as shown in Figure 3.1(a). In the following, these two contributions are discussed.

The SAM file format is an ASCII-based tab delimited text format to represent DNA sequence data as shown in Figure 3.1(c). Its in-memory SAM representation is a columnar format that consists of the same fields (columns) used in SAM to store the corresponding sequence mapping data as shown in Figure 3.1(d). The Arrow frameworks requires defining the data types for each field in a schema stored as part of the data object, as shown in Figure 3.1(e). The schema defines the ArrowSAM data format as listed more explicitly in Table 3.1.

Table 3.1 ArrowSAM Schema

Index	Field	Datatype
0	QNAME	String
1	FLAG	Int32
2	RNAME	Int32
3	POS	Int32
4	MAPQ	Int32
5	CIGAR	String
6	RNEXT	Int32
7	PNEXT	Int32
8	TLEN	Int32
9	SEQ	String
10	QUAL	String
11	TAG	String

Arrow stores the columnar data fields in contiguous memory chunks in so-called RecordBatches as shown in Figure 3.1(b). Each RecordBatch is a combination of a schema, which specifies the types of data fields of the ArrowSAM record, the data itself, in addition to some meta data.

3.1.3. Implementation

BWA-MEM integration: BWA-MEM aligns the raw read sequences against a large reference genome such as that of a human. We used ArrowSAM to store the mapping data produced by BWA-MEM from query and reference genome files. We modified BWA-MEM to use Arrow libraries to write each chromosome (1-22, X, Y and M) sequence mapping data in a separate Arrow RecordBatch. At the end of the alignment process, all the RecordBatches are assigned to a shared memory pool of plasma objects. Each plasma object has its own identifications (objectID). Tools that need to use the data generated by BWA-MEM can access this data managed by plasma through zero-copy shared memory access [10]. Doing so enables other tools to access all shared RecordBatches in parallel.

Sorting through pandas dataframes: Pandas is a powerful and easy to use Python library, which provides data structures, data cleaning and analysis tools. Dataframes is an in-memory data library that provides structures to store different types of data in tabular format to perform operations on the data in columns/rows. Any row in a dataframe can be accessed with its index, while a column can be accessed by its name. A column can also be a series in pandas. Using dataframes illustrates the powerful capabilities of in-memory data representation. First of all, dataframes is able to sort the chromosomes in parallel using pandas built-in sorting function with Python Arrow bindings (PyArrow) while accessing data residing in-memory, which takes place across two applications written in different languages (one in C and the other in Python). Secondly, tools like Picard and Sambamba are used to sort the SAM file according to the

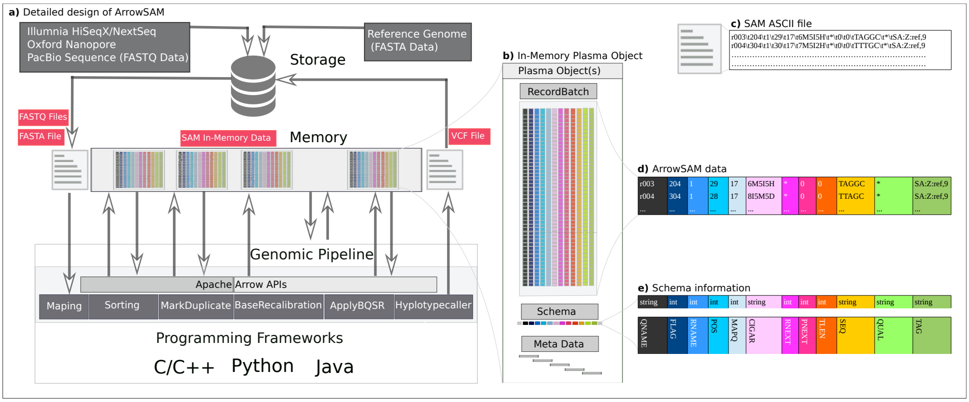

a) Detailed design of ArrowSAM
Illumnia HiSeqX/NextSeq
Oxford Nanopore
PacBio Sequence (FASTQ Data)
Reference Genome
(FASTA Data)
Storage
FASTQ Files
FASTA File
SAM In-Memory Data
Memory
VCF File
Genomic Pipeline
Apache Arrow APIs
Maping
Sorting
MarkDuplicate
BaseRecalibration
ApplyBQSR
Hyplotypecaller
Programming Frameworks
C/C++
Python
Java
b) In-Memory Plasma Object
Plasma Object(s)
RecordBatch
Schema
Meta Data
c) SAM ASCII file
r003\t204\t1\t29\t17\t6M5I5H\t*\t0\t0\tTAGGC\t*\tSA:Z:ref,9
r004\t304\t1\t30\t17\t7M5I2H\t*\t0\t0\tTTTGC\t*\tSA:Z:ref,9
d) ArrowSAM data
r003 204 1 29 17 6M5I5H * 0 0 TAGGC * SA:Z:ref,9
r004 304 1 28 17 8I5M5D * 0 0 TTAGC * SA:Z:ref,9
e) Schema information
string int int int int string int int int string string string
QNAME FLAG RNAME POS MAPQ CIGAR RNEXT PNEXT TLEN SEQ QUAL TAG

Figure 3.1 a) Genomics pipeline using ArrowSAM format for all intermediate steps to allow in-memory intermediate data storage, which means I/O disk access is only needed to load data into memory at the beginning and to write data back to disk at the end. b) Arrow RecordBatch enclosed in plasma object store. c) SAM file in ASCII text format. d) SAM file in RecordBatch format. e) Schema specifies the data types of ArrowSAM.

chromosome name and start positions of each chromosome. This type of sorting becomes computationally intensive when the whole SAM file needs to be parsed and sorted based on the values stored in only two fields of that file. This can be parallelized and made more efficient in our approach. Using pandas dataframes, sorting each individual chromosome is performed based on the start position of reads in that particular chromosome. All the RecordBatches are fed to pandas dataframes to sort all the chromosomes in parallel. After sorting, the sorted chromosomes RecordBatches are assigned to plasma shared memory again for subsequent applications to access.

Picard MarkDuplicate integration: After sorting the chromosomes data by their coordinates, the duplicate reads with low quality should be removed. The Picard MarkDuplicate tool is considered as a benchmark for this task. This tool reads the SAM file two times, first when building the sorted read end lists and then removing marked duplicates from the file. To overcome this I/O overhead, we just read the data as ArrowSAM format in-memory once, accessing only five fields (QNAME, FLAG, RNAME, POS, CIGAR and RNEXT) needed to perform the MarkDuplicate operation. We modified htsjdk (a java API used in Picard and many other tools for managing I/O access of DNA sequencing data files) and MarkDuplicate to read data from all RecordBatches in parallel from plasma shared memory. Our implementation processes this data in parallel in Picard and writes back the updated FLAG field in ArrowSAM which sets duplicate bit. After finishing this process, the shared memory plasma objects are available for further variant calling processes for in-memory and parallel execution.

3.1.4. Evaluation

This section evaluates the speedup and efficiency achieved using ArrowSAM for pre-processing of sequence data while mapping, sorting and marking duplicates against existing frameworks.

Experimental setup

All the experiments and comparisons are performed on a dual socket Intel Xeon server with E5-2680 v4 CPU running at 2.4 GHz. A total of 192 GB of DDR4 DRAM with maximum of 76.8 GB/s bandwidth is available for whole system.

We use Illumina HiSeq generated NA12878 dataset of whole exome sequencing (WES) of human with 30x sequencing coverage with paired-end reads and a read length of 100 bps. Similarly for whole genome sequencing (WGS), we use Illumina HiSeq generated NA12878 dataset sample SRR622461 with sequencing coverage of 2x with paired-end reads and a read length of 100 bps. Human Genome Reference, Build 37 (GRCh37/hg19) is used as a reference genome for all workflows in our experiments for both WES and WGS.

The code and scripts for running all workflows is freely available at: https://github.com/abs-tudelft/ArrowSAM. Tools and libraries and their version numbers used in our experiments are listed in Table 3.2.

Table 3.2 Tools and libraries used in the experimental setup

Tools/APIs	Version
BWA-MEM [5]	0.7.17
Picard [6]	2.18.14
Sambamba [7]	v0.6.8
elPrep [11]	v4.1.5
Arrow C/C++/Java [8]	0.11.0
PyArrow [12]	0.11.0
Plasma object store [9]	0.11.0

Performance evaluation

In this section, we compare our approach with state-of-the-art tools and approaches used for pre-processing of genomics sequencing data. All speedups are compared for best performance scenarios.

Picard tools are considered as benchmarks in genome analysis pipelines, such as Picard MarkDuplicate. This tool was adapted to use our ArrowSAM in-memory format. The MarkDuplicate process is compute intensive but there is a significant amount of time (approximately 30%) spent in I/O operations. Picard uses htsjdk as a base library to read/write SAM files. We modified this tool from two perspectives:

- Instead of reading from and writing to files, it now reads/writes from in-memory RecordBatches, using only those fields/columns necessary for

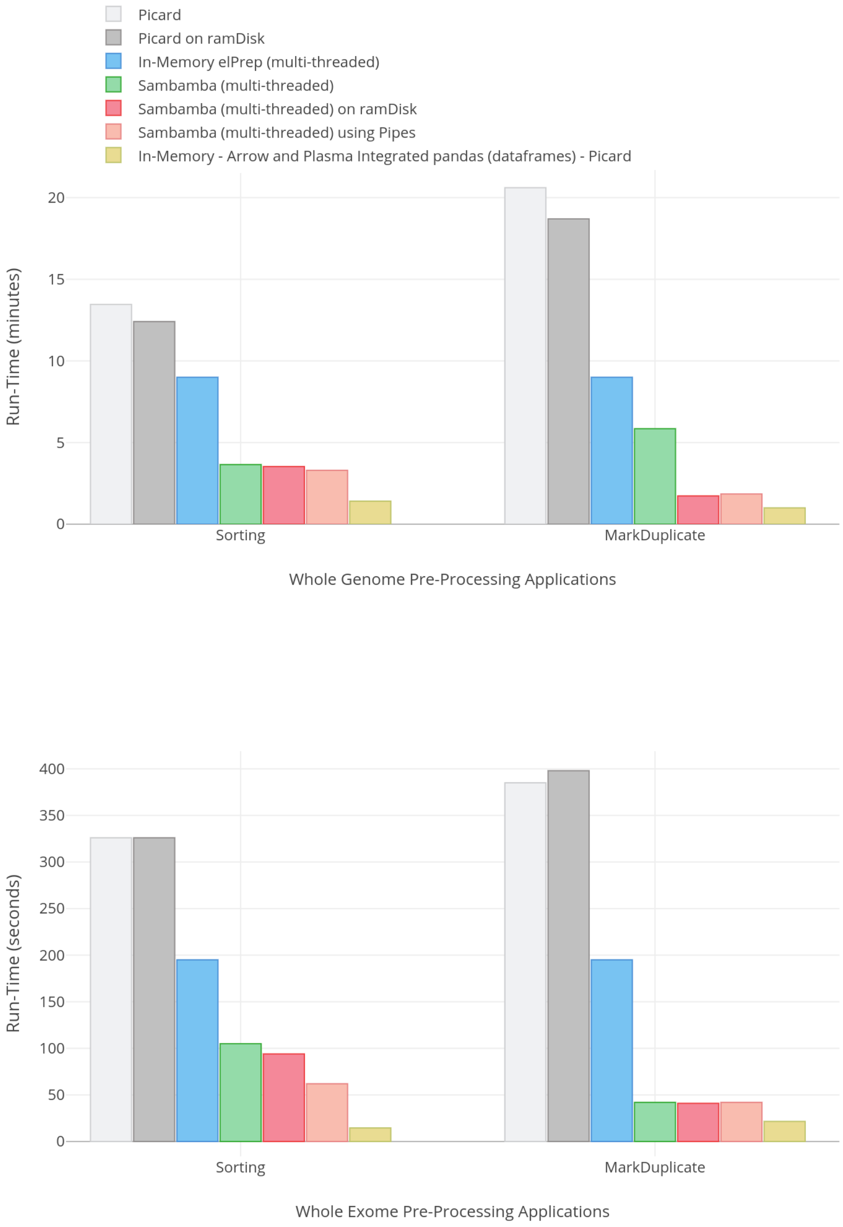


Figure 3.2 Execution time of Picard, Sambamba, elPrep and ArrowSAM based sorting and MarkDuplicate for whole genome (top) and whole exome (bottom) data sets.

MarkDuplicate operations.

- Picard is single threaded. We changed it to be multi-threaded so that each thread can operate on a separate chromosome data set.

The first modification provides the benefit of using only the required fields to perform MarkDuplicate operations instead of parsing all the reads in a SAM files. As shown in Figure 3.2, our implementation gives 8x and 21x speedups on Picard sorting for genome and exome data sets, respectively. Similarly for Picard MarkDuplicate, we achieve 21x and 18x speed-ups on for genome and exome data sets, respectively.

Sambamba is a multi-threaded tool to manipulate SAM files for pre-processing steps in genomics pipelines. This tool gives close to linear speedup for up to 8 threads but adding more threads provides diminishing returns in performance on a multi-core system. The main reason behind this is the file system itself. The I/O communication gets saturated by initiating more threads and CPU performance also degrades because of cache contention [7]. As shown in Figure 3.2, our implementation gives 2x speedup on Sambamba sorting for both genome and exome data sets. Similarly for Sambamba MarkDup, we achieve 1.8x and 3x speedups for genome and exome data sets, respectively.

elPrep is the latest set of multi-threaded tools for pre-processing SAM files in-memory. We have also tested and compared these tools with our implementation for pre-processing applications and results show that our implementation gives more than 5x speedup over elPrep. elPrep performs sorting and mark duplicate in a single command, with a total run-time for both stages equally divided in run-time graphs Figure 3.2. Samblaster is yet another tool used for pre-processing SAM files, which is faster than Sambamba, but has not been considered for performance comparison here because it produces a different output for the MarkDuplicate stage than Picard.

3.1.5. Discussion

CPU usage: Figure 3.3 shows CPU utilization for standard Picard (left) as well as ArrowSAM-based (bottom) sorting and MarkDuplicate for whole exome data. In both sorting and duplicates removal stages, the parallelization offered by shared memory plasma objects results in a large speedup. All 25 chromosomes are sorted and duplicates are removed in parallel. In Picard sorting the CPU utilization is poor and the tool is mostly waiting for I/O. The average CPU utilization is only 5%. The Picard MarkDuplicate also has very low CPU utilization, although better than sorting. On the other hand, the CPU utilization of our implementation, which uses pandas dataframes is much better than Picard sorting. The CPU utilization of MarkDuplicate in our implementation remains close to 95% during the whole execution stage. The improved CPU utilization is due to Arrow in-memory storage and parallel execution of processes, each working on a different chromosome.

Memory access: Picard tools read a whole line from the SAM file and then parse/extract all the fields from it. Using ArrowSAM, we only access those SAM

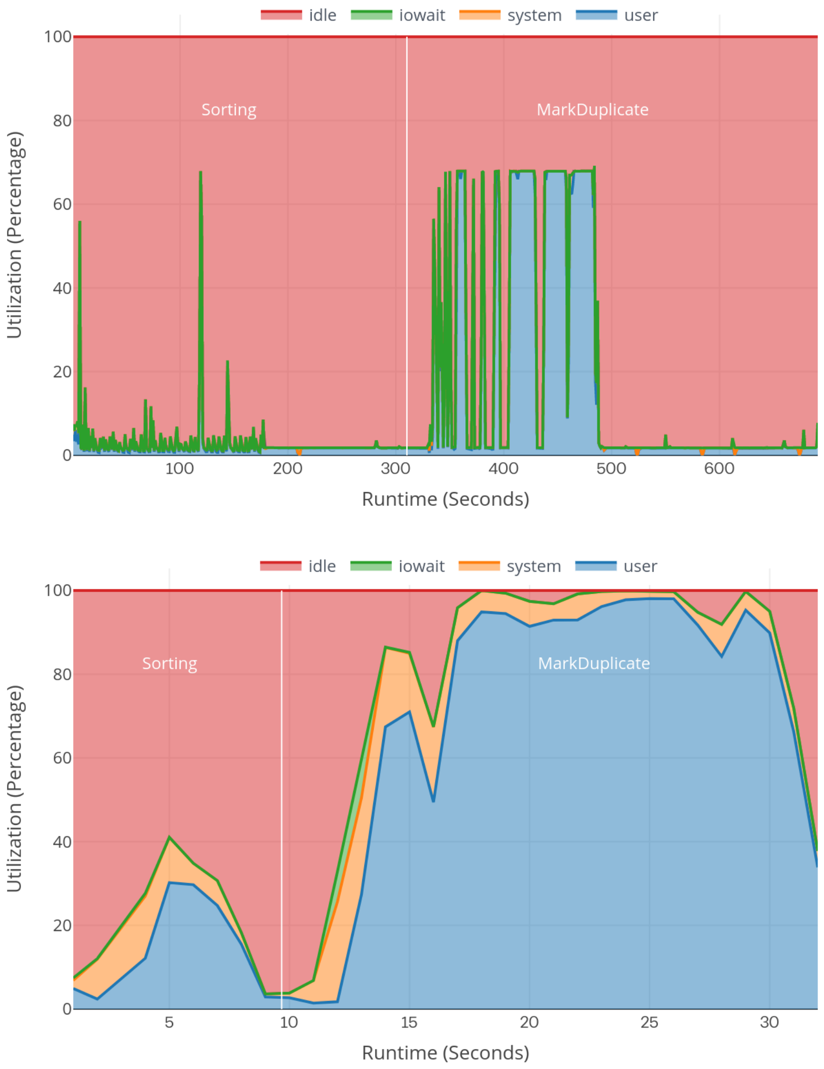


Figure 3.3 CPU resources utilization for standard Picard (top) as well as ArrowSAM-based (bottom) sorting and MarkDuplicate for whole exome data.

fields which are required by that specific tool to process. In addition, due to the columnar format of ArrowSAM, our implementation is able to better exploit cache locality. Figure 3.4 shows a comparison of level-1 (L1), level-2 (L2), and last-level cache (LLC) statistics for Picard as well as ArrowSAM-based sorting (left) and MarkDuplicate (right) applications for whole exome data set. The figure shows that, all levels of cache accesses decrease due to the fewer number of in-memory fields that need to be accessed for the sorting and marking duplicates processes in WES. Cache miss rate also decreases in all cache levels and particularly in L1 cache.

Memory usage: Unlike other tools, ArrowSAM data resides fully in-memory. Therefore, all the data is placed in a shared memory pool of plasma objects. After sorting, input plasma objects can be removed to free space for the new sorted data which is used in subsequent applications. Other than this, no additional memory is required for intermediate operations. elPrep is an alternative tool that also uses in-memory processing. Memory used by ArrowSAM and elPrep for WES and WGS data sets in pre-processing applications is shown in Table 3.5.

Table 3.3 Memory footprint for in-memory processing tools

Tool	Exome	Genome
elPrep	32GB	68GB
Arrow-based pandas and Picard	20GB	48GB

3.1.6. Related work

Many in-memory implementations of genomic variant discovery pipelines have been proposed. Almost all these implementations are cluster scaled and do not specifically exploit single node performance. Many use the input data parallelism to distribute the jobs on a cluster [13] and some of them take the benefit of the Apache Spark big data framework for in-memory data management [14, 15]. Our focus is to exploit performance of single node systems.

In addition, some research focuses on creating new genomics tools and algorithms that are more efficient than existing standard genomics pipelines [16]. ADAM [17], a set of formats and APIs uses Apache Avro and Parquet for storage and the Spark programming model for in-memory data caching to reduce the I/O overhead. The results show that ADAM is 3x slower than multi-threaded Sambamba in small number of cluster cores up to 64. In elPrep [11], the authors report 13x speedup over GATK best practices pipeline for whole-exome and 7.4x faster for whole-genome using maximum memory and storage footprints. The main drawback of these tools is lacking validation in the field which reduces their impact.

Other research focuses on innovative hardware platforms to execute genomics algorithms more efficiently [18]. In [19] a large pool of different types of memories are created and connected to processing resources through the Gen-Z communication protocol to investigate the concept of memory-driven computing. The

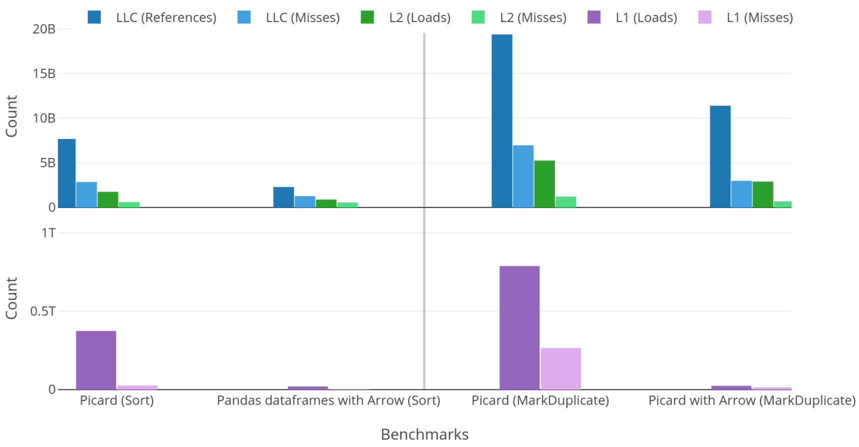


Figure 3.4 A comparison of level-1 (L1), level-2 (L2), and last-level cache (LLC) statistics for Picard as well as ArrowSAM-based sorting (left) and MarkDuplicate (right) applications for whole exome data set. (LLC ref. stands for LLC references)

memory is shared across running processes to avoid intermediate I/O operations. This systems also allows byte-addressability and load/store instructions to access memory. They reported 5.9x speedup on baseline implementation for some assembly algorithms, the source code is not available. Some researchers use high-performance hardware accelerators such as GPUs [20] and FPGAs [21] to accelerate computationally intensive parts of genomics pipelines, but availability of such accelerators in the field remains limited.

3.1.7. Conclusion

This paper proposed a new in-memory SAM data representation called ArrowSAM that makes use of the columnar in-memory capabilities of Apache Arrow. The paper showed the benefit of using ArrowSAM for genomic data storage and processing for genomics data pre-processing: mapping, sorting and mark duplicates. This allows us to process genomics data in-memory through shared memory plasma objects in parallel without the need for storing intermediate results through I/O into disk. Results show speedup of 28x for sorting and 15x for mark duplicates with respect to I/O based processing, more than 4x and 30% memory access reduction for sorting and mark duplicates, respectively, high CPU resources utilization, as well as better cache locality. These results indicate the potential of adopting a standard in-memory data format and shared memory objects for genomic pipeline processing. Future research will focus on extending our work for the complete genomics variant calling pipeline. In addition, we plan to integrate ArrowSAM into big data frameworks like Apache Spark to enable cluster scale scalability of genomics applications.

3.2. Performance optimization of GATK best practices pipeline

Immense improvements in sequencing technologies enable producing large amounts of high throughput and cost effective next-generation sequencing (NGS) data. This data needs to be processed efficiently for further downstream analyses. Computing systems need this large amounts of data closer to the processor (with low latency) for fast and efficient processing. However, existing workflows depend heavily on disk storage and access, to process this dataincurs huge disk I/O overheads. Previously, due to the cost, volatility and other physical constraints of DRAM memory, it was not feasible to place large amounts of working data sets in memory. However, recent developments in storage-class memory and non-volatile memory technologies have enabled computing systems to place huge data in memory to process it directly from memory to avoid disk I/O bottlenecks.

To exploit the benefits of such memory systems efficiently, proper formatted data placement in memory and its high throughput access is necessary by avoiding (de)-serialization and copy overheads in between processes. For this purpose, we use the newly developed Apache Arrow, a cross-language development framework that provides language-independent columnar in-memory data format for efficient in-memory big data analytics. This allows genomics applications developed in different programming languages to communicate in-memory without having to access disk storage and avoiding (de)-serialization and copy overheads. We integrate Apache Arrow in-memory based Sequence Alignment/Map (SAM) format and its shared memory objects store library in widely used genomics high throughput data processing applications like BWA-MEM, Picard and GATK to allow in-memory communication between these applications. In addition, this also allows us to exploit the cache locality of tabular data and parallel processing capabilities through shared memory objects.

Our implementation shows that adopting in-memory SAM representation in genomics high throughput data processing applications results in better system resource utilization, low number of memory accesses due to high cache locality exploitation and parallel scalability due to shared memory objects. Our implementation focuses on the GATK best practices recommended workflows for germline analysis on whole genome sequencing (WGS) and whole exome sequencing (WES) data sets. We compare a number of existing in-memory data placing and sharing techniques like ramDisk and Unix pipes to show how columnar in-memory data representation outperforms both. We achieve a speedup of 4.85x and 4.76x for WGS and WES data, respectively, in overall execution time of variant calling workflows. Similarly, a speedup of 1.45x and 1.27x for these data sets, respectively, is achieved, as compared to the second fastest workflow. In some individual tools, particularly in sorting, duplicates removal and base quality score recalibration the speedup is even more promising. The code and scripts used in our experiments are available in both container and repository form at: https://github.com/abs-tudelft/ArrowSAM

3.2.1. Introduction

The *genome* of an organism is the complete set of its genetic material represented by its DNA sequence. Each cell in a human body contains a complete replication of the approximately 3 billion base pairs (bps) of DNA. The *genomics* field emphasizes on the understanding of structure, mapping and function of individuals genes (the genome) to get insights into their interaction and evolution with respect to one's environment. In *comparative genomics*, complete genome features of different spices are extensively compared (for example with a reference genome) using computational tools. These comparisons can lead to fully characterize the resemblances and differences in one's genomic features, trace down their origin or lineage, how the change or loss emerges throughout the evolutionary lineages and discover ways to cure diseases caused by genetic variations and developing personalized medicine and improving environmental health [22].

Variant calling is indispensable for comparative genomics as it reveals deep insights into nucleotide-level organismal differences in some specific traits among populations from an individual genome sequence data. Variant calling discerns genetic variations in three categories like, single nucleotide polymorphisms (SNPs), insertions and deletions (indels), and/or structural variants (SVs, may also include Copy Number Variations (CNVs), duplication, translocation, etc). An *SNP* reports a single base change in two genomes while the DNA around that base remains unchanged. *Indels* are single bases which have been inserted, or deleted in a genome when aligning to another reference genome. *Structural variants* are observed in organism's chromosome structures. Generally defined as a region of DNA approximately 1 kilo base-pairs (kbp) or larger in size having variations in the form of inversions, translocations or deletions, insertions and CNVs (also called duplications). DNA sequencing reveals that CNVs are commonly observed in various organisms, particularly in human, which vary from individual to individual. Approximately two third of whole human genome is composed of such repeats.

DNA can mutate in any of the somatic cells or in germinal cells (germ cells); such variations are referred as somatic and germinal mutations, respectively. *Somatic analysis* identifies the variations in normal and tumor affected tissues. Somatic mutations/variations can cause cancer or other diseases. In *germline analysis* the variations in an individual's DNA inherited from parents are analyzed to identify presence of inherited disease.

In *whole-genome sequencing (WGS)* the complete set of DNA sequences (both the entire protein coding and the non-coding regions of the genome) of an organism are determined. This gives a comprehensive and precise fingerprint of the whole DNA. *Whole-exome sequencing (WES)* instead just focuses on collecting DNA sequences of some specific regions (like protein coding). WES samples are typically sequenced at 100X or 30X coverage which focuses on less than ~5% of the complete genome. Both techniques have their own benefits. WES saves costs and also gives more DNA coverage resulting in higher accuracy. WGS covers the complete genome which is good for fully characterizing and understanding the genome.

Genome/Exome pre-processing

The pre-processed genomics data can be used for timely identification of gene mutation, diagnosis of disease as well as the development of targeted therapies. Generally pre-processing steps include alignment, sorting and duplicate reads removal from target genome sequence data. Many tools have been developed for analysis of high throughput sequencing data, from local alignment database search tools like BLAST [23], FASTA [24] to pairwise alignment tools like MALIGN [25], EMBOSS [26], tools like BLAT [27] Bowtie2 [28] and BWA-MEM [29] for short read sequence alignments and Minimap and Miniasm [30], DALIGNER [31] and DARWIN [32] tools for long reads alignment and mapping. Tools like SAMtools [33], Picard [6], Sambamba [7] and samblaster [34] are developed for alignment post-processing stages like indexing, sorting, duplicates removal in SAM/BAM (Binary Alignment/Map) files.

Variant callers

GATK and FreeBayes are commonly used open-source tools for germline variant calling analysis. Tools like VarScan [35], VarDict [36], MuTect2 [37] are used for somatic variant calling analysis. FreeBayes [38], SNVer [39] and LoFreq [40] are also used for both germline and somatic variant calling analysis. Pisces [41] and Strelka2 [42] are recently developed open source tools by Illumina for short variant calling to analyze both germline and somatic variations. DeepVariant [43] is deep convolutional neural network based variant caller.Both Strelka2 and DeepVariant variant callers outperform GATK, FreeBayes and SAMtools in PrecisionFDA (pFDA) Challenges for precision and accuracy on indels and SNVs for different data sets. The output of these tools is generated in the variant calling format (VCF) to visualize and further analyze the detected variations.

Challenges in genomics data processing

Comparative genomics is a young field. To process and analyze genomics data, the research community is actively working to develop new, efficient and optimized algorithms, techniques and tools, usually programmed in a variety of languages, such as C, Java or Python. As we have mentioned earlier, in order to construct a whole workflow for complete genome analysis, one has to use a combination of different open-source tools. These tools share the following common characteristics that impose limitations on the performance achievable by the genomics workflow.

- These tools are developed to use traditional I/O file systems, which incur a huge I/O bottleneck in computation due to disk bandwidth [4]. Each tool reads from the I/O disks, computes and writes back to disk.
- Due to the virtualized nature of some popular languages used to develop genomics tools (such as Java and Python), these tools cannot exploit mod-

ern hardware features like multi-core parallelization, Single instruction, multiple data (SIMD) vectorization and accelerators (like GPU or FPGAs) performance very well.

- In between processes data communication developed in different languages, a huge (de)-serialization and copy overheads incur.

Motivation

New storage-class memory (SCM) technologies will soon replace the existing long latency and block-based data transfer HDDs/SSDs storage. Intel's phase-change memory (PCM) based Optane DC (Data Center) Persistent Memory is one of the first candidates in this paradigm to accelerate big data workloads for in-memory analytics and provide fast startup-times for legacy applications/virtual machines in cloud environments [44–46]. Using these memories to store SAM data in columnar format and shared memory objects can provide benefit in many aspects to improve overall system throughout:

- One is related to the tabular nature of genomics data (SAM) in-memory.
- Second is related to underlying hardware technology to exploit the maximum cache spatial locality and SIMD vectorization capabilities of modern multi-core systems.
- Third is to avoid (de)serialization of data when processing in different languages. Shared memory objects of SAM data can be processed in parallel.

We use DRAM as an alternative to such memory technologies for evaluation purpose because of its same characteristics of byte-addressability (load/store access to memory) and lower latency.

3.2.2. Background

This section provides a short description of widely-adopted GATK variant calling workflow, NGS technologies and the amount of data they produce and the challenges in processing this data. A brief introduction to the Apache Arrow framework and its Plasma shared memory API is also given.

Genome Analysis Toolkit (GATK): GATK [47] from the Broad Institute is considered as a benchmark for variant calling discovery.As the SAM [48] is a de-facto format for storing NGS data, its compressed and indexed BAM [48] version is used in GATK tools as input file(s). GATK tools produce variant calling outputs in many different formats like VCF, GVCF and different useful statistics in text format. GATK internal architecture is based on the philosophy of MapReduce [49] functional programming paradigm to achieve maximum parallel efficiency by distributing data among processes. In MapReduce programming, the computations are accomplished in two steps; first the problem is divided into many discrete independent tasks which are fed to the map function. After completion of tasks their respective outputs are merged into a reduce function to generate a final output product. GATK reads/writes data files through htsjdk library, divides and prepares data in traversals then processes in walker modules. The walker modules provide the map and reduce functions for data consumption [50].

GATK best practices workflows: For the analysis and interpretation of NGS data to be used in clinical settings, different tools and workflows have been created. GATK recommended best practices for variant calling proposes BWA-MEM for mapping reads, while Picard or Sambamba can be used for sorting and mark duplicates removal in the reads. Base Quality Score Recalibration (BQSR) in GATK adjusts the quality score of reads by employing machine learning algorithm. The following common GATK workflows [51, 52] are available in GATK4 for different types of variant calling. 1. For identifying germline short variants (SNPs and indels) in one or more individuals the Haplotypecaller algorithm is used to generate a joint callset in VCF format. 2. Similarly MuTect2 is used for somatic short variants (SNVs and indels) identification in one or more tumor samples in a single individual, with or without a matched normal sample. 3. For germline short variants (SNPs and indels) discovery in human exome sequencing data the workflow uses intervals file in BED format while the Haplotypecaller algorithm is used to generate a joint callset in VCF format.

Next-generation sequencing: technologies and data: The first ever Human Genome Project [1990—2003] concluded an initial sequence draft of human genome consisting of approximately 2.85 billion nucleotides [53]. Since then, genomics data has been increasing rapidly due to the innovations in genome sequencing technologies and analysis methods. Second Generation Sequencing (NGS) technologies like Illumina’s HiSeqX and NextSeq produce whole genome, high throughput and high quality short read data at a total cost of $1K per genome, which is expected to drop down below $100 for more advanced sequencing technologies. Third generation sequencing technologies are now capable of sequencing reads of more than 10 kbp in length, such as Oxford Nanopore, Single Molecule Real-Time and Pacific BioSciences sequencing technologies. The ongoing pace of these technologies promises even more longer reads of ~100 kbp on average. Long reads produced by third generation sequencing technologies provide the prospect to fully characterize genomes at high resolution for precision medicine [3].

Apache Arrow: The Apache Arrow [8] project was initiated by the Apache Foundation in 2016. This framework provides an open and a common standardized format for different programming languages for reading/writing tabular data in-memory. Through language-specific libraries, multiple languages can share data without any copying or serialization. This in-memory access of data through Apache Arrow is illustrated in Figure 3.5. At the time of writing, Apache Arrow supports the following languages: Go, C, C++, C#, Java, JavaScript, R, Rust, MATLAB, Ruby and Python. Interfaces exist for GPGPU programming, through Arrow CUDA interfaces. External tools to support FPGA accelerators also exist through the Fletcher project [54]. In the Arrow format, data entries (records) are stored in a table called a RecordBatch. Each record field is stored in a separate column of the RecordBatch table in a manner that is as contiguous as possible in memory. This is called an Arrow Array which can store data of different types—i.e. int, float, strings, binary, timestamps and lists, but also nested types (such as lists of lists, etc.). Arrays may have different types of physical buffers to store data. This layout provides higher spatial locality when iterating over column contiguous data entries for better CPU cache performance. SIMD (Single instruction, multiple data) vector operations can also benefit from such a layout as vector elements are already aligned properly in memory.

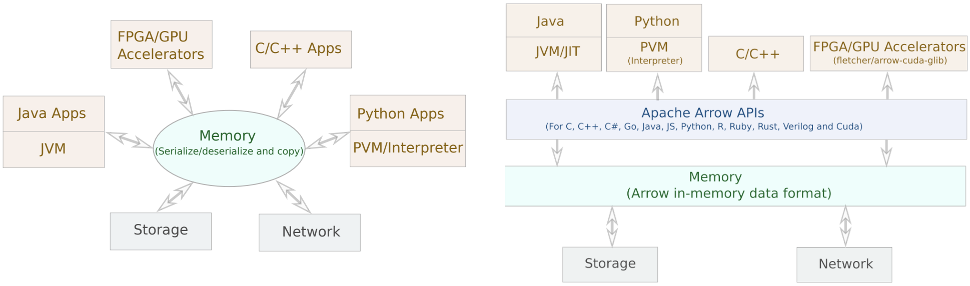


Figure 3.5 Left: An example where (de)serialization and copy takes place when data is exchanged between different languages and platforms. **Right:** Apache Arrow provides a unified in-memory format for data placement which can be used in many languages and platforms avoiding the (de)serialization and copy overhead.

Plasma in-memory object store: Plasma is an inter-process communication (IPC) component of Arrow, that handles shared memory pools across different heterogeneous systems [9]. To perform IPC, processes can create Plasma objects inside the shared memory pool, that are typically data buffers underlying an Arrow RecordBatch. Through the shared memory pool, Plasma enables zero-copy data sharing between processes.

3.2.3. Implementation

In order to enable genome pre-processing applications and GATK to use in-memory SAM data, two main optimization are required. First, we need to define an in-memory Arrow representation of the SAM data. Second, the applications need to be adapted to access the new in-memory SAM data as shown in Figure 3.6. These applications access, update and create new data fields as shown in Figure 3.7. In the following, these two optimizations are discussed.

In-memory SAM format

The SAM file format is an ASCII based, tab delimited text format to represent DNA sequence data. We create an in-memory SAM representation using the Apache Arrow columnar format that consists of the same fields (columns) used in SAM to store the corresponding data, this format is also explicitly explained in our previous work [55]. We call this the ArrowSAM format, this stores the data in Record-

Batches. Each RecordBatch is a combination of a schema, which specifies the types of data fields of ArrowSAM and the data itself, more details of in-memory Arrow data representation can be found in [55].

Genomics applications can use ArrowSAM to create RecordBatches of genomics data in-memory.

RecordBatch columnar data can be deleted/updated in the same application but to make data usable in other applications we have to use shared memory flat buffers.

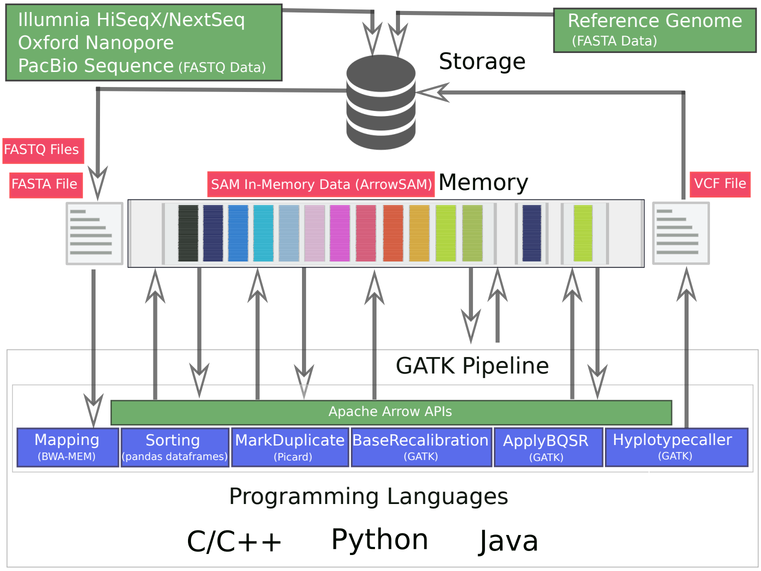


Figure 3.6 In-memory architecture of GATK best practices recommended workflow using Arrow in-memory SAM representation for all intermediate steps.

BWA-MEM integration: BWA-MEM is the most popular alignment algorithms in the bioinformatics community due to its efficient and accurate alignment of raw FASTQ data against a large reference genome. After performing alignment of each read, it creates a SAM record of twelve data fields as shown in Figure 3.7. Instead of writing these records in a SAM file, we modified BWA-MEM to use the ArrowSAM format and Arrow libraries to store these records in Arrow Buffers. We have created as many such buffers as number of chromosomes. So we check the reference name (RNAME) of each record and insert to its respective buffer. At the end of the alignment process, all the buffers are converted to RecordBatches which are inserted into shared memory pool.

Sorting through Pandas Dataframes: Randomly generated SAM reads need to be sorted by their respective chromosome and individual coordinates (begin positions) within a chromosome. Pandas is a powerful and easy to use python library, which provides data structures, data cleaning and analysis tools. Dataframes is an in-memory data library that provides structures to store different types of data in tabular format to perform operations on the data in columns/rows. Any row in a dataframe can be accessed with its index, while a column can be accessed by its name. A column can also be a series in pandas. Using dataframes with python arrow bindings (PyArrow) illustrates the powerful capabilities of in-memory data representation. Tools like Picard, SAMtools and Sambamba are used to sort the reads in a SAM file according to the chromosome name and start positions of each read. This type of sorting becomes computationally intensive when the whole SAM file needs to be parsed and sorted based on these two fields. In contrast, our implementation uses pandas dataframes to sort each individual chromosome based on the start position of reads in that particular chromosome. This reduces the computational effort needed to sort the reads since we already assign them to the RecordBatch that belongs to their own chromosome. Therefore, we only need to sort them based on their position. We create new RecordBatches for each chromosome with sorted data as shown in Figure 3.7.

All shared memory objects of chromosomes are fed to pandas dataframes to sort in parallel. After sorting, the new sorted chromosomes RecordBatches are stored in shared memory by deleting previous shared memory objects, to be used by subsequent applications.

Picard MarkDuplicate integration: After sorting the reads by their coordinates, the duplicate reads with low quality should be removed. The MarkDuplicate tool in the Picard package is considered as a standard algorithm for duplicate reads removal. This tool reads the SAM files two times, first when building the sorted read end lists and second when removing marked duplicates from those lists by comparing each individual read in the file. This tool has two main limitations: first it reads SAM data sequentially from the input file, second it converts all input file reads data into their corresponding SAM records. SAM files are usually stored in disk in compressed format (called BAM) that has a compression ratio of about 30%. This means every time we read and write these files to disk, we have to incur a the overhead of compression and decompression. To overcome these overheads, we just read the data as ArrowSAM format in-memory once, accessing only five fields `(QNAME, FLAG, RNAME, POS, CIGAR and RNEXT)` as shown in Figure 3.7, which are actually needed to perform the MarkDuplicate operation. For this purpose, We have modified the htsjdk (a java API used in Picard, GATK and many other tools for managing I/O access of high-throughput sequencing data files) to access shared memory stored plasma objects, and parse them to their respective RecordBatch. Each SAM read with the above mentioned five fields is accessed via index. Each shared memory object contains one chromosome SAM data. To take advantage of this, our implementation processes all chromosomes in parallel by initiating as many Picard instances as number of chromosomes. After processing reads, MarkDuplicate sets the duplicate bit in the `FLAG` field, so only the `FLAG` field is updated in this process which is written in a separate shared memory object for each chromosome. After completion of the MarkDuplicate stage, the sorted and updated duplicate flag data is available in shared memory objects for further analysis.

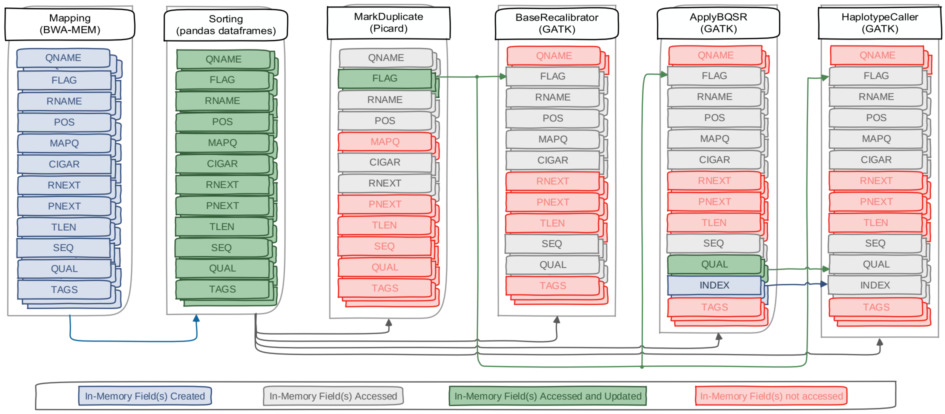


Figure 3.7 In-memory SAM data placement for all chromosomes (1-22, X, Y and M) in GATK best practices workflow. Applications access it through shared memory plasma objects. For higher sequencing coverage data i.e. WGS data, each chromosome with size more than 2GB is further divided for scalability.

GATK BaseRecalibration integration: Variant calling heavily relies on the assigned base quality scores per base in individual reads. These scores are estimates of sequencing machine errors in producing bases. However, these scores are also affected due to systematic errors in the sequencing machines. BaseRecalibration finds systematic error patterns by analyzing how these errors vary over all bases. Only seven fields are accessed: six fields (RNAME, POS, MAPQ, CIGAR, SEQ and QUAL) from ArrowSAM records of shared memory objects created in the 'Sorting' process, and one (FLAG) field created in MarkDuplicate process as shown in Figure 3.7. We have also modified the access to the htsjdk library for this application similar to the MarkDuplicate application. All shared memory objects of individual chromosomes are processed in parallel by initiating as many as BaseRecalibration instances as number of chromosomes. All relevant information generated by this tool is recorded in tables.

GATK ApplyBQSR integration: ApplyBQSR applies numerical corrections to each individual base call based on the patterns identified in BaseRecalibration tables. This application generates new QUAL and INDEX fields which are written in a separate shared memory object. In this application, the same seven fields are accessed: six fields (RNAME, POS, MAPQ, CIGAR, SEQ and QUAL) from in-memory SAM records of shared memory objects created in 'Sorting' process and one (FLAG) field created in MarkDuplicate process as shown in Figure 3.7. We have also modified htsjdk library for this process similar to previous processes except in generating output. Because we limit our processing to specific parts of the genome by filtering out unused intervals as provided in a special filtering file (called bed file), only those reads which fall in these specific intervals are forwarded for further processing. To properly map the newly created QUAL field output with that of original in-memory ArrowSAM data of the sorting process (to be used in the next application), we have appended an additional field 'index'. This field stores the index of the original read. All shared memory objects of individual chromosomes are processed in parallel by initiating as many as ApplyBQSR instances as the number of chromosomes.

GATK Haplotypecaller integration: Haplotypecaller calls SNPs and indels through local de-novo assembly in active regions. Active regions are those which have some sufficient probability of variation. Here eight fields are accessed: five fields (RNAME, POS, MAPQ, CIGAR and SEQ) from ArrowSAM records of shared memory objects created in the 'Sorting' application, one (FLAG) field created in the MarkDuplicate application and two (QUAL and INDEX) field created in the ApplyBQSR application as shown in Figure 3.7. We have also modified access to the htsjdk library for this process similar to previous processes like ApplyBQSR. First, the INDEX field is checked as an alternative to intervals, so that particular index in the original ArrowSAM object created in the 'Sorting' process is accessed. This indexing technique has one benefit and also one drawback. In terms of benefit, using the index field, we access only those fields which fall in given bed file intervals for exome analysis. Drawback is related to cache performance. Due to repeatedly changing the index during reads access, the cache spatial locality cannot be exploited efficiently. The output of this process is generated in VCF file format. Because we are processing all the shared memory objects of individual chromosomes in parallel, separate chromosome files are generated which need to be merged for further variants analysis.

3.2.4. Methods

We compare our ArrowSAM-based workflow to a number of popular workflows used in the field. For alignment we use BWA-MEM for all workflows due to its high accuracy and efficiency. For sorting and duplicate removal, Picard, Sambamba and elPrep (sfm) have been used. GATK and elPrep are used for the base recalibration and variant discovery stages. The reason behind selecting elPrep for performance comparison is the fact that it uses in-memory, and multi-threading techniques for pre-processing and variant discovery, while reporting to produce the same accuracy as that of GATK [11]. In contrast, our implementation also facilitates in-memory and multi-threading features while using the exact same Picard and GATK applications. The reason for selecting Sambamba for the comparison is its multi-threaded nature and for being more efficient than other open source tools available for sorting and mark duplicate operations with the same accuracy as Picard.

In the following subsections, we discuss the workflows used in comparison to ArrowSAM based implementation.

Storage (BWA-MEM - Picard - GATK)

This combination of tools is used in almost all GATK recommended best practices workflows for both whole genome and whole exome sequencing analysis. Both reference and query raw data sets are placed in local storage and all applications access data through local disk I/O. All the immediate results of each application are also written in local disk in standard SAM/BAM files.

Storage (BWA-MEM - Sambamba - GATK)

Sambamba is used here as an alternative to Picard for sorting and mark duplicates operations. Sambamba as mentioned earlier is faster than Picard for both of these applications because of multi-threading. But unfortunately parallel performance of Sambamba is limited and not scalable due to I/O saturation. All data sets and immediate results of each application are using local storage for I/O.

ramDisk (BWA-MEM - Sambamba - GATK)

In this workflow, we use ramDisk (memory-mapped disk) instead of local storage, since we can improve performance of these applications by placing data closer to the processor. This way, all data sets and immediate results of each application are kept in ramDisk in standard SAM/BAM files.

ramDisk (BWA-MEM - Sambamba - GATK (Parallel))

We can use some sort of naive parallelism for performance improvement in some GATK applications. For example in whole exome sequencing, BaseRecalibration application uses an interval file with -L option. If we split the interval file for each chromosome and pass the individual interval files to multiple instances of the BaseRecalibration application each executed for an individual chromosome in parallel, it will generate output 'tables' separate for each chromosome. Then, ApplyBQSR can also use the individual chromosomes interval files and 'tables'. So running the ApplyBQSR instances in parallel will generate new BAM files for each individual chromosome separately. These individual chromosome BAMs and interval files can be passed to parallel instances of Haplotypecallers, which will generate separate VCF files for individual chromosomes. These VCF files can then be merged in GATK.

ramDisk (BWA-MEM - Sambamba (Pipes) - GATK (Parallel))

We can use Unix pipes in some intermediate applications to redirect their standard output to other application in the workflow as their input to save the I/O time and disk resources of local storage. Using Unix pipes, the output of an application is not stored in disk, but is buffered in memory temporarily until it is consumed by the next application in the pipe. We also naive parallelism for performance improvement in some GATK applications as mentioned in above method.

ArrowSAM

This is our implementation proposed by this paper which uses in-memory ArrowSAM format and shared memory plasma objects to exploit cache spatial locality and multi-core efficiency. 1) Alignment is done in BWA-MEM which has already multi-threading support and output ArrowSAM data is placed in shared memory objects in respective chromosomes boundaries, 2) followed by coordinates based sorting using Plasma dataframes on all chromosomes (chromosomes greater than 2GB in WGS data sets are further divided with zero-copy overhead) to run sorting algorithm in parallel. 3) Picard MarkDuplicate is then run on resulting shared memory data chunks in parallel creating new FLAG field in-memory. 4) GATK BaseRecalibration generates tables for all ArrowSAM data chunks in parallel, 5) ApplyBQSR creates QUAL field by running parallel on all ArrowSAM data chunks and finally Haplotypecaller generates separate VCF files for each chromosome/chunk (in case chromosome greater than 2GB). These files are merged to generate a final VCF using GATK for further analysis.

elPrep

As discussed earlier, elPrep [11] is a multi-threaded pre-processing tool to operate on SAM/BAM data in-memory. In this tool, sorting, duplicate marking and base quality score recalibration algorithms are optimized for parallel execution. This tool has two runtime options, one is sfm, which uses less memory as compare to the other one called filter, which uses a large memory pool for in-memory processing. As reported in the paper, this tool has the same accuracy for pre-processing of SAM/BAM data as that of the GATK recommended best practices workflow. Therefore, we included this workflow for speedup comparison with all other workflows as this tool is more closely related to our implementation in the context of multi-threading and in-memory data placement and execution. Finally, GATK Haplotypecaller is used for variant calling.

3.2.5. Results

For evaluation of our in-memory SAM format and Apache Arrow integration into BWA-MEM, Picard and GATK tools, we have created a number of different workflows using state-of-the-art tools and techniques in accordance with GATK best practices workflow for whole genome and exome sequencing. We have run all these workflows with their recommended settings. In the "Performance evaluation" section below we describe the measured performance, while we discuss the results in the "Discussion" section.

The individual applications execution times of the various workflows for WES are shown in Figure 3.8 while Figure 3.9 shows the execution times for individual application for WGS. Similarly, the total execution times of the workflows are shown in Figure 3.10 and Figure 3.11 for WES and WGS data sets, respectively.

Performance evaluation

In this section, we compare execution time of our GATK recommended best practices variant calling workflow using ArrowSAM with other state-of-the-art workflows as discussed in the "Methods" section on high throughput genome and exome data sets.

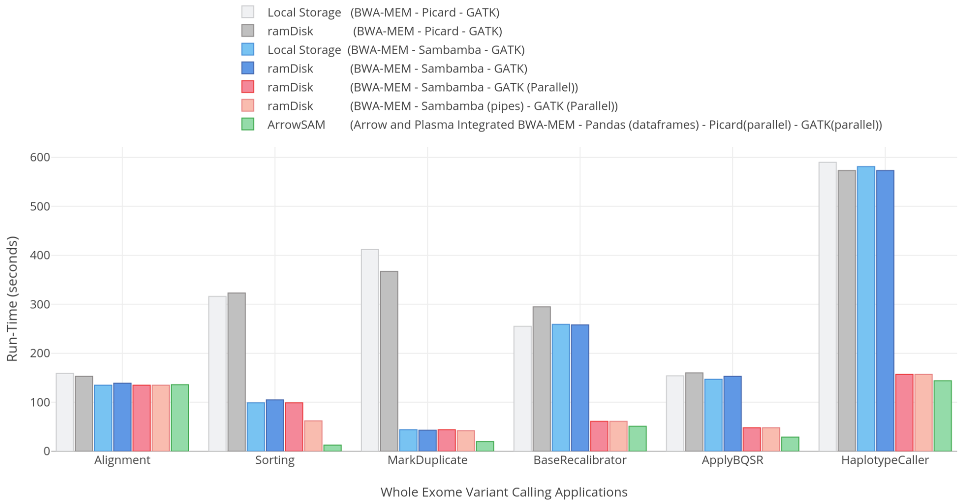


Figure 3.8 Runtimes (in seconds) of individual variant calling applications on whole exome data set using different workflow options (i.e. ramDisk, Pipes for Sambamba and chromosome wise parallelism in GATK).

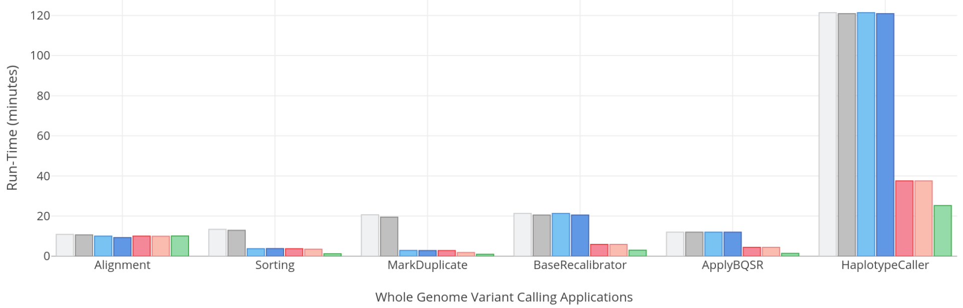


Figure 3.9 Runtimes (in minutes) of individual variant calling applications on whole genome data set using different workflow options (i.e. ramDisk, Pipes for Sambamba and chromosome wise parallelism in GATK).

Storage (BWA-MEM - Picard - GATK)

This represents the *baseline* workflow. The main performance bottleneck in this workflow is single-threaded disk I/O access of SAM/BAM file(s) by the htsjdk library which is used in Picard and GATK tools. This workflow takes highest runtime from pre-processing to variant calling among all workflows.

Storage (BWA-MEM - Sambamba - GATK)

Replacing Picard with Sambamba for sorting and duplicate removal gives significant speedup in overall workflow execution as shown in Figure 3.8 and 3.9. But Sambamba's performance does not scale very well since increasing the number of threads above 12 for sorting and mark duplicate gives no performance improvement. Still, the individual time of sorting and mark duplicate is significantly less than the baseline with 12 threads resulting in an overall execution time speedup of 1.5x as compared to the baseline for whole workflow.

ramDisk (BWA-MEM - Sambamba - GATK)

ramDisk is frequently suggested as an alternative for fast processing. We have observed that for GATK, only a small performance improvement is achieved. In the case of using ramDisk with Sambamba, there is even a reduction in performance for WES.

ramDisk (BWA-MEM - Sambamba - GATK (Parallel))

Using ramDisk for Sambamba while running GATK in parallel for all chromosomes is able to achieve a better performance improvement as compared to all previous workflows. The speedup is 3.5x and 3x for WES and WGS data as compared to the baseline workflow, respectively.

ramDisk (BWA-MEM - Sambamba (Pipes) - GATK (Parallel))

Redirecting output of BWA-MEM to Sambamba using Unix pipes slightly improves the performance of ramDisk. This is the best possible scenario of performance improvement as compared to previous workflows. It gives an overall speedup of 3.7x and 3.1x for WES and WGS data over the baseline workflow, respectively.

ArrowSAM

ArrowSAM based workflow is the fastest among all workflows. This workflow is made as scalable as possible by employing widely used pre-processing and variant calling algorithms of Picard and GATK. We achieve a speedup of 4.76x and 4.85x for WES and WGS data in overall execution time as compared to baseline workflow, respectively. Compared to the fastest parallelized workflow (ramDisk (BWA-MEM - Sambamba (Pipes) - GATK (Parallel))), our ArrowSAM workflow achieves a speedup of 1.27x and 1.45x with WES and WGS data, respectively.

elPrep

elPrep is a single application that can be used as a plug-in replacement for all pre-processing tools. That is the reason individual applications runtime is not shown in Figure 3.8 and Figure 3.9.

When using the sfm option, elPrep gives a speedup of 2.49x and 3.26x over the baseline for WES and WGS data, respectively. However, it is 1.91x and 1.45x slower than ArrowSAM for WES and WGS data, respectively. This tool also has a filter option, which gives 4.7x speedup over the baseline and is only slightly slower than ArrowSAM for WES data at the expense of using 4x more memory than ArrowSAM. We are not able to run the filter option on WGS data due to large memory requirement. In the elPrep paper [11], the authors also do not show the results with the filter option for WGS data.

Evaluation system

All experiments and comparisons are performed on a dual socket Intel Xeon Server with E5-2680 v4 CPU running at 2.40GHz. Each processor has 14 physical cores with support of 28 hyper-threading jobs. Both processors are connected through Intel QuickPath Interconnect and share memory through non-uniform memory access architecture. A total of 192-GBytes of DDR4 DRAM with a maximum of 76.8 GB/s bandwidth is available for the whole system. A local storage of 1-TBytes and the same amount of network attached storage is available on the system. CentOS 7.3 Minimal Server operating system is installed. All workflows are executed through bash scripts.

Tools: The Apache Arrow framework and all its related libraries (like cglib, PyArrow and arrow-java) are installed in a Singularity container for ease of use to external users. The installed tools are listed in Table 3.4 with their versions for future reference.

Table 3.4 Tools and libraries used in the experimental setup

Tools/APIs	Version
BWA-MEM [29]	0.7.17
Picard [6]	2.18.14
GATK [47]	4.0.12.0
Sambamba [7]	0.6.8
elPrep [11]	4.1.5
Arrow C/C++/Java [8]	0.11.0
PyArrow [12]	0.11.0
Plasma Object Store [9]	0.11.0

Dataset: We use Illumina HiSeq generated NA12878 dataset [56] with paired-end reads of WES of human with 30x sequencing coverage. Similarly for WGS, we use Illumina HiSeq generated NA12878 dataset sample SRR622461 with paired-end reads with sequencing coverage of 6x (we further lower the coverage to 2x due to memory limit on our evaluation system). Read length of 100 base-pairs is used for all data. Genome Human Genome Reference, Build 37 (GRCh37/hg19) is used as a reference genome. All workflows in our experiments use this data set for both WES and WGS.

Table 3.5 Peak memory usage for in-memory processing tools.

Tool	Exome	Genome
elPrep (sfm)	6.8GB	68GB
elPrep (filter)	26GB	X
ArrowSAM	7.2GB	69GB

Memory footprint: Our implementation is solely memory based, so all the data between BWA-MEM and Haplotypecaller applications remains in memory. We only compare runtime peak memory utilization of elPrep and ArrowSAM since in-memory resource requirements vary for intermediate operations. Table 3.5 lists the memory usage for both tools. elPrep uses almost the same memory as ArrowSAM on WES data with the sfm option enabled and uses 4x more memory with the filter option. For WGS data elPrep (sfm) and ArrowSAM have the same memory footprint but elPrep (filter) memory footprint is not available due to the large memory requirements beyond available system memory resources. This use case is also not covered in the original elPrep paper. The results show that ArrowSAM only uses memory that is comparable to the size of the SAM file for both WES and WGS data sets.

Discussion

Here we discuss some characteristics and limitations of our implementation in context of future perspective of in-memory data formats and processing for variant calling applications.

Parallelization and scalability: In ArrowSAM, all applications are capable to process data in parallel. The chunks of SAM data can be based on chromosomes or on the required data size. So that the memory plasma objects can be shared between different applications, which results in a large speedup in the overall runtime of individual applications.

CPU utilization: Depending on data partition in ArrowSAM, the maximum number of CPUs can be used for processing data in individual applications.

Cache locality: Due to the in-memory columnar data format, our implementation is able to exploit cache locality efficiently. All levels of cache accesses decrease in the sorting and mark duplicate applications, particularly due to the fewer number of in-memory fields (mostly integer type) access as discussed in [55]. Cache miss rate also decreases in all cache levels and particularly in level-1 cache. In BaseRecalibration, ApplyBQSR and Haplotypecaller applications we also exploit cache locality but it is not much significant as compared to previous applications because of two reasons, 1. algorithms for these tools are not developed in such a way to exploit cache locality efficiently, and 2. base sequences (`SEQ`) and qualities (`QUAL`) fields are also being accessed which pollute cache lines early in these applications.

Accuracy: We did not change any part of actual algorithms in all Picard and GATK applications. Therefore, our results are exactly the same as in the original implementation of both tools.

Code and scripts availability

The code for the in-memory ArrowSAM representation, all related Apache Arrow libraries for C, Java and Python languages and plasma shared memory process are installed on a singularity container which is is freely available at:
`https://github.com/abs-tudelft/ArrowSAM`

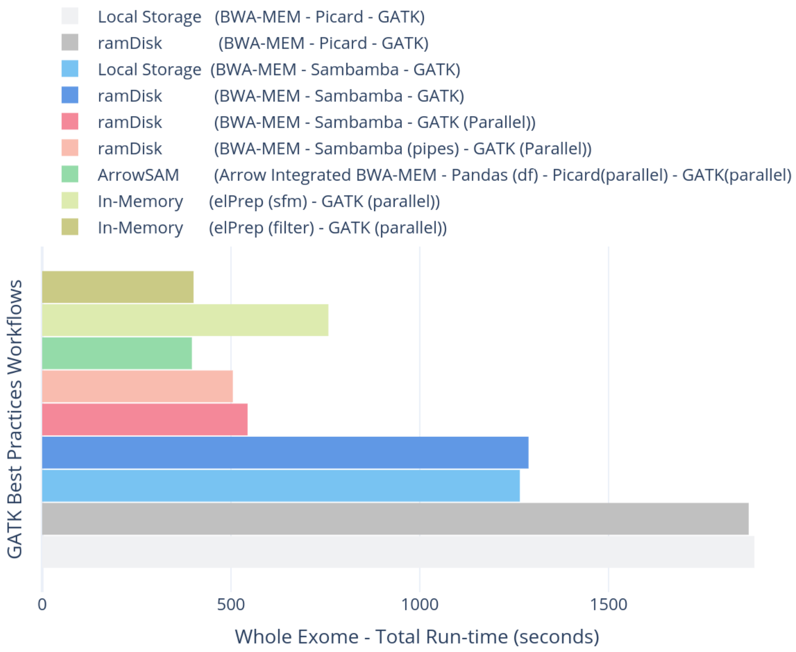


Figure 3.10 Total runtimes (in seconds) for complete variant calling workflows using different efficient options (i.e. ramDisk, Pipes for Sambamba and chromosome wise parallelism in GATK) on whole exome data set.

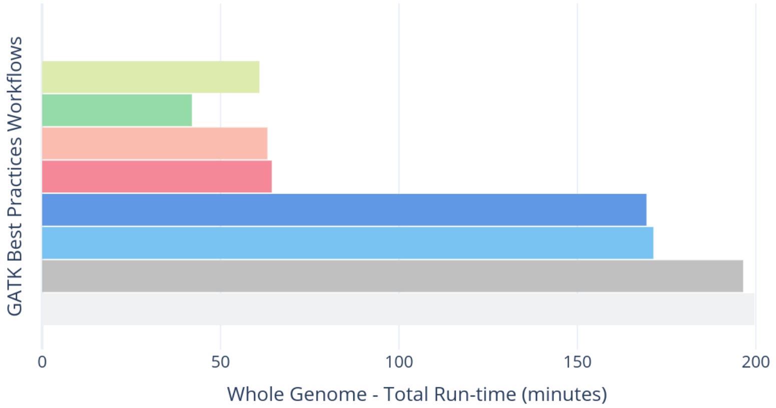


Figure 3.11 Total runtimes (in minutes) for complete variant calling workflows using different efficient options (i.e. ramDisk, Pipes for Sambamba and chromosome wise parallelism in GATK) on whole genome data set.

3.2.6. Related work

Many in-memory implementations of genomics variant discovery workflows have been presented in the literature. Many of these implementations are cluster scaled and do not exploit single node performance taking advantage of the Apache Spark framework [57] for in-memory data management like SparkGA [15] and ADAM [17]. These implementations are not discussed in this paper. Our focus is to exploit the performance of single node systems.

Aginome IMP Platform [58], a GPU based sequence analysis tool set which uses in-memory database to store intermediate results for further analysis. To use in-memory database, the IMP Platform modifies FreeBayes [38] and GATK for use in variant calling. IMP with GATK speeds up the variant detection workflow by 30x, while IMP with FreeBayes improves the performance by 100x as compared to the BWA-GATK workflow. However, this tool is not open-sourced. The Sentieon Genomics Tools [59], report 10x performance improvement over GATK, MuTect and MuTect2 workflows by eliminating intermediate files merging. This tool also reports improving the performance of BWA-MEM by 1.9x times. However, this tool is also not available publicly and the paper does not discuss the details of the implementation.

elPrep [60], is a set of tools for pre-processing SAM/BAM files for variant calling. It is a multi-threaded, single command plug-in replacement tool which processes the data in-memory instead of reading and writing to I/O for each operation. In elPrep 4 [11], the authors reported 13x speedup over GATK best practices workflow for whole-exome and 7.4x speedup for whole-genome data using maximum memory and storage footprints, at the expense of excessive memory utilization. They also compare the results for a cluster deployment to show the scalability for high performance computing infrastructure. In memory-driven computing, a large pool of different types of memories are created and connected to the processing resources through the Gen-Z communication protocol. The memory is shared across the processes being executed to avoid intermediate I/O operations. This systems also allows byte-addressability and load/store instructions to access memory. [19] used a Gen-Z enabled platform for genomics and reported 5.9x speedup over the SAMtools baseline implementation for a number of DNA assembly algorithms. The source code is not available.

Some researchers use high-performance hardware accelerators such as GPUs [61] and FPGAs [62] to accelerate computationally intensive parts of genomics pipelines, but availability of such accelerators in the field remains limited.

3.2.7. Conclusion

In this work, we integrate our Apache Arrow in-memory SAM representation (ArrowSAM) into genomics pre-processing and variant calling applications.

Our implementation shows that adopting in-memory SAM representation in genomics high throughput data processing applications results in better system resource utilization, low number of memory accesses due to high cache locality exploitation and parallel scalability due to shared memory objects. We compare a number of existing in-memory data placing and sharing techniques like ramDisk and Unix pipes to show how columnar in-memory data representation outperforms both. We achieve a speedup of 4.85x and 4.76x for WGS and WES data sets in overall execution time of variant calling workflows, respectively. Similarly, a speedup of 1.45x and 1.27x for these data sets is achieved, as compared to the second fastest workflow.

In future work, to feed the processor fast and properly formatted data, in-memory data management techniques will be explored more rigorously to leverage the benefits of modern hardware features like multi-cores, vector units and to exploit cache locality in the presence of persistent memory technologies. We also plan to use ArrowSAM in big data frameworks like Spark for cluster level scalability of genomics applications.

REFERENCES

[1] T. Ahmad, N. Ahmed, J. Peltenburg, and Z. Al-Ars. “ArrowSAM: In-Memory Genomics Data Processing Using Apache Arrow”. In: *2020 3rd International Conference on Computer Applications Information Security (ICCAIS)*. 2020, pp. 1–6. doi: 10.1109/ICCAIS48893.2020.9096725.

[2] T. Ahmad, N. Ahmed, Z. Al-Ars, and H. P. Hofstee. “Optimizing performance of GATK workflows using Apache Arrow In-Memory data framework”. In: *BMC Genomics* 21.10 (Nov. 2020), p. 683. issn: 1471-2164. doi: 10.1186/s12864-020-07013-y. url: https://doi.org/10.1186/s12864-020-07013-y.

[3] D. Gurdasani, M. S. Sandhu, T. Porter, M. O. Pollard, and A. J. Mentzer. “Long reads: their purpose and place”. In: *Human Molecular Genetics* 27.R2 (May 2018), R234–R241. issn: 0964-6906. doi: 10.1093/hmg/ddy177. eprint: http://oup.prod.sis.lan/hmg/article-pdf/27/R2/R234/25229925/ddy177.pdf. url: https://doi.org/10.1093/hmg/ddy177.

[4] Y. Diao, A. Roy, and T. Bloom. *Building Highly-Optimized, Low-Latency Pipelines for Genomic Data Analysis*.

[5] H. Li. *Aligning sequence reads, clone sequences and assembly contigs with BWA-MEM*. 2013. eprint: arXiv:1303.3997.

[6] *Picard toolkit*. http://broadinstitute.github.io/picard/. 2019.

[7] A. Tarasov, A. J. Vilella, E. Cuppen, I. J. Nijman, and P. Prins. “Sambamba: fast processing of NGS alignment formats”. In: *Bioinformatics* 31.12 (June 2015). 25697820[pmid], pp. 2032–2034. issn: 1367-4811. doi: 10.1093/bioinformatics/btv098. url: https://www.ncbi.nlm.nih.gov/pubmed/25697820.

[8] Apache. *Apache Arrow: A cross-language development platform for in-memory data [Accessed 29th Dec. 2019]*. 2019. url: https://arrow.apache.org/.

[9] Apache. *Plasma In-Memory Object Store [Accessed 29th Dec. 2019]*. 2019. url: https://arrow.apache.org/blog/2017/08/08/plasma-in-memory-object-store/.

[10] U. L. Technology. *Apache Arrow Platform*. 2019. url: https://ursalabs.org/tech/.

[11] C. Herzeel, P. Costanza, D. Decap, J. Fostier, and W. Verachtert. “elPrep 4: A multithreaded framework for sequence analysis”. In: *PLOS ONE* 14.2 (Feb. 2019). Ed. by L. Chen, e0209523. doi: 10.1371/journal.pone.0209523. url: https://doi.org/10.1371/journal.pone.0209523.

[12] ApacheFoundation. *Python library for Apache Arrow*. 2019. url: https://pypi.org/project/pyarrow/.

[13] B. Institute. *Introduction to the GATK Best Practices*. 2019. url: https://software.broadinstitute.org/gatk/best-practices/.

[14] H. Mushtaq and Z. Al-Ars. “Cluster-based Apache Spark implementation of the GATK DNA analysis pipeline”. In: *Proceedings of the IEEE International Conference on Bioinformatics and Biomedicine*. Washington, DC, USA, 2015, pp. 1471–1477. doi: 10.1109/BIBM.2015.7359893. url: https://ieeexplore.ieee.org/abstract/document/7359893.

[15] H. Mushtaq, F. Liu, C. Costa, G. Liu, P. Hofstee, and Z. Al-Ars. “SparkGA: A Spark Framework for Cost Effective, Fast and Accurate DNA Analysis at Scale”. In: *Proceedings of the 8th ACM International Conference on Bioinformatics, Computational Biology,and Health Informatics*. ACM-BCB ’17. Boston, Massachusetts, USA: ACM, 2017, pp. 148–157. isbn: 978-1-4503-4722-8. doi: 10.1145/3107411.3107438. url: http://doi.acm.org/10.1145/3107411.3107438.

[16] L. Hasan and Z. Al-Ars. “An Efficient and High Performance Linear Recursive Variable Expansion Implementation of the Smith-Waterman Algorithm”. In: *Proceedings of the IEEE Engineering in Medicine and Biology Conference*. Minneapolis, MN, USA, 2009, pp. 3845–3848. doi: 10.1109/IEMBS.2009.5332567. url: https://ieeexplore.ieee.org/abstract/document/5332567.

[17] M. Massie, F. Nothaft, C. Hartl, C. Kozanitis, A. Schumacher, A. D. Joseph, and D. A. Patterson. *ADAM: Genomics Formats and Processing Patterns for Cloud Scale Computing*. Tech. rep. UCB/EECS-2013-207, EECS Department, University of California, Berkeley, 2013.

[18] L. Hasan and Z. Al-Ars. “An Overview of Hardware-based Acceleration of Biological Sequence Alignment”. In: *Computational Biology and Applied Bioinformatics*. InTech, 2011, pp. 187–202.

[19] M. Becker, M. Chabbi, S. Warnat-Herresthal, K. Klee, J. Schulte-Schrepping, P. Biernat, P. Guenther, K. Bassler, R. Craig, H. Schultze, S. Singhal, T. Ulas, and J. L. Schultze. “Memory-driven computing accelerates genomic data processing”. In: (Jan. 2019). doi: 10.1101/519579. url: https://doi.org/10.1101/519579.

[20] E. Houtgast, V. Sima, K. Bertels, and Z. Al-Ars. “GPU-Accelerated BWA-MEM Genomic Mapping Algorithm Using Adaptive Load Balancing”. In: *Architecture of Computing Systems (ARCS)*. Nuremberg, Germany: Springer, 2016, pp. 130–142. doi: 10.1007/978-3-319-30695-7_10. url: https:

//link.springer.com/chapter/10.1007/978-3-319-30695-7%5C_10.

[21] J. Peltenburg, S. Ren, K. Bertels, and Z. Al-Ars. “Maximizing systolic array efficiency to accelerate the PairHMM Forward Algorithm”. In: *IEEE International Conference on Bioinformatics and Biomedicine*. Shenzhen, China, 2016, pp. 758–762. doi: 10.1109/BIBM.2016.7822616. url: https://ieeexplore.ieee.org/abstract/document/7822616.

[22] X. Xia. *Comparative Genomics*. June 2013. isbn: 978-3-642-37146-2. doi: 10.1007/978-3-642-37146-2.

[23] S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman. “Basic local alignment search tool”. In: *Journal of Molecular Biology* 215.3 (1990), pp. 403–410. issn: 0022-2836. doi: https://doi.org/10.1016/S0022-2836(05)80360-2. url: http://www.sciencedirect.com/science/article/pii/S0022283605803602.

[24] D. J Lipman and W. Pearson. “Rapid and Sensitive Protein Similarity Searches”. In: *Science (New York, N.Y.)* 227 (Apr. 1985), pp. 1435–41. doi: 10.1126/science.2983426.

[25] W. Wheeler and D. S. Gladstein. “MALIGN: A multiple sequence alignment program”. In: *Journal of Heredity* 85 (Sept. 1994). doi: 10.1093/oxfordjournals.jhered.a111492.

[26] P. Rice, I. Longden, and A. Bleasby. “EMBOSS: The European molecular biology open software suite”. In: *Trends in genetics : TIG* 16 (July 2000), pp. 276–7. doi: 10.1016/S0168-9525(00)02024-2.

[27] W. James Kent. “BLAT - The BLAST-like alignment tool”. In: *Genome research* 12 (May 2002), pp. 656–64. doi: 10.1101/gr.229202..

[28] B. Langmead and S. L. Salzberg. “Fast gapped-read alignment with Bowtie 2”. In: *Nature Methods* 9.4 (Apr. 2012), pp. 357–359. issn: 1548-7105. doi: 10.1038/nmeth.1923. url: https://doi.org/10.1038/nmeth.1923.

[29] H. Li and R. Durbin. “Fast and accurate short read alignment with Burrows–Wheeler transform”. In: *Bioinformatics* 25.14 (May 2009), pp. 1754–1760. issn: 1367-4803. doi: 10.1093/bioinformatics/btp324. url: https://doi.org/10.1093/bioinformatics/btp324.

[30] H. Li. “Minimap and miniasm: fast mapping and de novo assembly for noisy long sequences”. In: *Bioinformatics* 32.14 (July 2016). 27153593[pmid], pp. 2103–2110. issn: 1367-4811. doi: 10.1093/bioinformatics/btw152. url: https://www.ncbi.nlm.nih.gov/pubmed/27153593.

[31] G. Myers. “Efficient Local Alignment Discovery amongst Noisy Long Reads”. In: vol. 8701. Sept. 2014, pp. 52–67. doi: 10.1007/978-3-662-44753-6_5.

[32] Y. Turakhia, G. Bejerano, and W. J. Dally. “Darwin: A Genomics Co-Processor Provides up to 15,000X Acceleration on Long Read Assembly”. In: *SIGPLAN Not.* 53.2 (Mar. 2018), pp. 199–213. issn: 0362-1340. doi: 10.1145/3296957.3173193. url: https://doi.org/10.1145/3296957.3173193.

[33] H. Li. “The Sequence Alignment/Map format and SAMtools”. In: *Bioinformatics* 25 (Jan. 2009), pp. 2078–2079.

[34] G. G. Faust and I. M. Hall. “SAMBLASTER: fast duplicate marking and structural variant read extraction”. In: *Bioinformatics* 30.17 (Sept. 2014). 24812344[pmid], pp. 2503–2505. issn: 1367-4811. doi: 10.1093/bioinformatics/btu314. url: https://www.ncbi.nlm.nih.gov/pubmed/24812344.

[35] D. C. Koboldt, Q. Zhang, D. E. Larson, D. Shen, M. D. McLellan, L. Lin, C. A. Miller, E. R. Mardis, L. Ding, and R. K. Wilson. “VarScan 2: Somatic mutation and copy number alteration discovery in cancer by exome sequencing”. In: *Genome Research* 22.3 (Feb. 2012), pp. 568–576. doi: 10.1101/gr.129684.111. url: https://doi.org/10.1101/gr.129684.111.

[36] Z. Lai, A. Markovets, M. Ahdesmaki, B. Chapman, O. Hofmann, R. McEwen, J. Johnson, B. Dougherty, J. C. Barrett, and J. R. Dry. “VarDict: a novel and versatile variant caller for next-generation sequencing in cancer research”. In: *Nucleic Acids Research* 44.11 (Apr. 2016), e108–e108. doi: 10.1093/nar/gkw227. url: https://doi.org/10.1093/nar/gkw227.

[37] K. Cibulskis, M. S. Lawrence, S. L. Carter, A. Sivachenko, D. Jaffe, C. Sougnez, S. Gabriel, M. Meyerson, E. S. Lander, and G. Getz. “Sensitive detection of somatic point mutations in impure and heterogeneous cancer samples”. In: *Nature Biotechnology* 31 (Feb. 2013), 213 EP -. url: https://doi.org/10.1038/nbt.2514.

[38] E. Garrison and G. Marth. *Haplotype-based variant detection from short-read sequencing*. 2012. eprint: arXiv:1207.3907,[Accessed:11April2019].

[39] Z. Wei, W. Wang, P. Hu, G. J. Lyon, and H. Hakonarson. “SNVer: a statistical tool for variant calling in analysis of pooled or individual next-generation sequencing data”. In: *Nucleic Acids Research* 39.19 (Aug. 2011), e132–e132. doi: 10.1093/nar/gkr599. url: https://doi.org/10.1093/nar/gkr599.

[40] A. Wilm, P. P. K. Aw, D. Bertrand, G. H. T. Yeo, S. H. Ong, C. H. Wong, C. C. Khor, R. Petric, M. L. Hibberd, and N. Nagarajan. “LoFreq: a sequence-quality aware, ultra-sensitive variant caller for uncovering cell-population heterogeneity from high-throughput sequencing datasets”. In: *Nucleic Acids Research* 40.22 (Oct. 2012), pp. 11189–11201. doi: 10.1093/nar/gks918. url: https://doi.org/10.1093/nar/gks918.

[41] T. Dunn, G. Berry, D. Emig-Agius, Y. Jiang, S. Lei, A. Iyer, N. Udar, H.-Y. Chuang, J. Hegarty, M. Dickover, B. Klotzle, J. Robbins, M. Bibikova, M. Peeters, and M. Strömberg. “Pisces: an accurate and versatile variant caller for somatic and germline next-generation sequencing data”. In: *Bioinformatics* 35.9 (Oct. 2018). Ed. by R. Schwartz, pp. 1579–1581. doi: 10.1093/bioinformatics/bty849. url: https://doi.org/10.1093/bioinformatics/bty849.

[42] S. Kim, K. Scheffler, A. L. Halpern, M. A. Bekritsky, E. Noh, M. Källberg, X. Chen, Y. Kim, D. Beyter, P. Krusche, and C. T. Saunders. “Strelka2: fast and accurate calling of germline and somatic variants”. In: *Nature Methods* 15.8 (2018), pp. 591–594. issn: 1548-7105. doi: 10.1038/s41592-018-0051-x. url: https://doi.org/10.1038/s41592-018-0051-x.

[43] R. Poplin, P.-C. Chang, D. Alexander, S. Schwartz, T. Colthurst, A. Ku, D. Newburger, J. Dijamco, N. Nguyen, P. T. Afshar, S. S. Gross, L. Dorfman, C. Y. McLean, and M. A. DePristo. “A universal SNP and small-indel variant caller using deep neural networks”. In: *Nature Biotechnology* 36 (Sept. 2018). url: https://doi.org/10.1038/nbt.4235.

[44] H. .-. P. Wong, S. Raoux, S. Kim, J. Liang, J. P. Reifenberg, B. Rajendran, M. Asheghi, and K. E. Goodson. “Phase Change Memory”. In: *Proceedings of the IEEE* 98.12 (Dec. 2010), pp. 2201–2227. issn: 0018-9219. doi: 10.1109/JPROC.2010.2070050.

[45] G. Burr, M. J. Breitwisch, M. Franceschini, D. Garetto, K. Gopalakrishnan, B. Jackson, B. Kurdi, C. Lam, L. A. Lastras, A. Padilla, B. Rajendran, S. Raoux, and R. S. Shenoy. “Phase Change Memory Technology”. In: *Journal of vacuum science and technology. B, Microelectronics and nanometer structures: processing, measurement, and phenomena: an official journal of the American Vacuum Society* 28 (Jan. 2010). doi: 10.1116/1.3301579.

[46] J. Condit, E. B. Nightingale, C. Frost, E. Ipek, B. Lee, D. Burger, and D. Coetzee. “Better I/O Through Byte-addressable, Persistent Memory”. In: *Proceedings of the ACM SIGOPS 22Nd Symposium on Operating Systems Principles*. SOSP ’09. Big Sky, Montana, USA: ACM, 2009, pp. 133–146. isbn: 978-1-60558-752-3. doi: 10.1145/1629575.1629589. url: http://doi.acm.org/10.1145/1629575.1629589.

[47] B. Institute. *Genome Analysis Toolkit [Accessed: 11 April 2019]*. 2010. url: https://software.broadinstitute.org/gatk/.

[48] T. S. F. S. W. Group. *Sequence Alignment/Map Format Specification [Accessed: 11 April 2019]*. 2010. url: samtools.github.io/hts-specs/SAMv1.pdf.

[49] J. Dean and S. Ghemawat. “MapReduce: Simplified Data Processing on Large Clusters”. In: *Commun. ACM* 51.1 (Jan. 2008), pp. 107–113. issn: 0001-0782. doi: 10.1145/1327452.1327492. url: http://doi.acm.org/10.1145/1327452.1327492.

[50] A. McKenna, M. Hanna, E. Banks, A. Sivachenko, K. Cibulskis, A. Kernytsky, K. Garimella, D. Altshuler, S. Gabriel, M. Daly, and M. A. DePristo. "The Genome Analysis Toolkit: a MapReduce framework for analyzing next-generation DNA sequencing data". In: *Genome research* (2010). doi: 10.1101/gr.107524.110. url: https://www.ncbi.nlm.nih.gov/pubmed/20644199.

[51] B. Institute. *GATK Best Practices Workflows [Accessed 11 April 2019]*. 2010. url: https://github.com/gatk-workflows.

[52] B. Institute. *GATK Variant Calling Pipelines*. 2010. url: https://software.broadinstitute.org/gatk/best-practices/.

[53] I. H. G. S. Consortium. "Finishing the euchromatic sequence of the human genome". In: *Nature* 431.7011 (2004), pp. 931–945. issn: 1476-4687. doi: 10.1038/nature03001. url: https://doi.org/10.1038/nature03001.

[54] J. Peltenburg, J. van Straten, M. Brobbel, H. P. Hofstee, and Z. Al-Ars. "Supporting Columnar In-memory Formats on FPGA: The Hardware Design of Fletcher for Apache Arrow". In: *Applied Reconfigurable Computing*. Ed. by C. Hochberger, B. Nelson, A. Koch, R. Woods, and P. Diniz. Cham: Springer International Publishing, 2019, pp. 32–47. isbn: 978-3-030-17227-5.

[55] T. Ahmad, J. Peltenburg, N. Ahmed, and Z. Al-Ars. "ArrowSAM: In-Memory Genomics Data Processing through Apache Arrow Framework". In: (2019). doi: 10.1101/741843.

[56] Illumina. *Illumina Cambridge Ltd. [Accessed 24th May 2019]*. 2012. url: ftp://ftp.1000genomes.ebi.ac.uk/vol1/ftp/phase3/data/NA12878/sequence_read/.

[57] Apache. *Apache Spark: Lightning-fast unified analytics engine [Accessed: 2nd April 2019]*. 2019. url: https://spark.apache.org/.

[58] S. Wang, W. Yang, X. Zhang, and R. Yu. "Performance Evaluation of IMP: A Rapid Secondary Analysis Pipeline for NGS Data". In: Dec. 2018, pp. 1170–1176. doi: 10.1109/BIBM.2018.8621573.

[59] D. N. Freed, R. Aldana, J. A. Weber, and J. S. Edwards. "The Sentieon Genomics Tools - A fast and accurate solution to variant calling from next-generation sequence data". In: (Mar. 2017). doi: 10.1101/115717. url: https://doi.org/10.1101/115717.

[60] C. Herzeel, P. Costanza, D. Decap, J. Fostier, and J. Reumers. "elPrep: High-Performance Preparation of Sequence Alignment/Map Files for Variant Calling". In: *PLOS ONE* 10.7 (July 2015). Ed. by C. Antoniewski, e0132868. doi: 10.1371/journal.pone.0132868. url: https://doi.org/10.1371/journal.pone.0132868.

[61] S. Ren, K. Bertels, and Z. Al-Ars. “Efficient Acceleration of the Pair-HMMs Forward Algorithm for GATK HaplotypeCaller on Graphics Processing Units”. In: *Evolutionary Bioinformatics* 14 (2018). doi: 10.1177/1176934318760543.

[62] E. J. Houtgast, V.-M. Sima, K. Bertels, and Z. Al-Ars. “Hardware acceleration of BWA-MEM genomic short read mapping for longer read lengths”. In: *Computational Biology and Chemistry* 75 (2018), pp. 54–64.

4

Apache Spark & Apache Arrow based variant calling

Prelude: *Based on the benefits of ArrowSAM integration into variant calling applications as discussed in the previous chapter, we widen the usage of this format for big data frameworks like Apache Spark. Apache Spark leverages the benefits of Apache Arrow for vectorized operations in user-defined functions using dataframes in PySpark. We integrate the DeepVariant-based whole variant calling pipeline including alignment, sorting, duplicate removal applications to process SAM data in-memory. We introduce a highly scalable chromosome region-specific data partitioning approach. We also compare both existing Apache Spark based variant calling workflows like ADAM and SparkGA2 with this implementation. The content of this chapter is based on our journal article entitled, "VC@Scale: Scalable and high-performance variant calling on cluster environments" [1] which was published in the Oxford GigaScience journal in 2021.*

4.1. Overview

In the past couple of years many new deep learning based variant calling methods like DeepVariant has emerged as more accurate method as compared to conventional variant calling algorithms like GATK HaplotypeCaller, Sterlka2, Freebayes albeit at higher computational costs. Therefore, there is a need for more scalable and higher performance workflows of these deep learning methods. Almost all existing cluster scaled variant calling workflows that use Apache Spark and Apache Hadoop as big data frameworks loosely integrate existing single node pre-processing and variant calling applications. Using Apache Spark just for distributing/scheduling data among loosely coupled applications or using I/O based storage for storing intermediate applications output does not exploit the full benefit of Apache Spark in-memory processing. In order to achieve this, we propose a native Spark-based workflow that uses Python and Apache Arrow to enable efficient transfer of data between different workflow stages. This benefits from the ease of programmability of Python and the high efficiency of Arrow's columnar in-memory data transformations.

Here we present a scalable, parallel and efficient implementation of next generation sequencing data pre-processing and variant calling workflows. Our design tightly integrates most pre-processing workflow stages, using Spark built-in functions to sort reads by coordinates, and mark duplicates efficiently. Our approach outperforms state-of-the-art implementations by more than 2x for the pre-processing stages, creating a scalable and high performance solution for DeepVariant for both CPU-only and CPU+GPU clusters.

We show the feasibility and easy scalability of our approach to achieve high performance and efficient resource utilization for variant calling analysis on HPC clusters using the standardized Apache Arrow data representations. All codes, scripts and configurations used to run our implementations are publicly available and open sourced:
https://github.com/abs-tudelft/variant-calling-at-scale

4.2. Introduction

Immense improvements in Next Generation Sequencing (NGS) technologies enable producing large amounts of high throughput and cost-effective raw genome datasets. On the one hand, this development paves the way to analyze more genomes with higher accuracy, but at the same time this creates the computational challenge of processing such a large amount of data in a timely fashion. The approximate raw data size of the human genome sequenced using NGS technologies is 300 GB when sequenced with 30x coverage, and can be more than 1 TB raw data with 300x sequencing coverage. The ongoing pace of development of these technologies promises even longer reads of up to 100 kbp and with more coverage depth.

To process and prepare raw data for downstream analysis, many open-source and proprietary bioinformatics tools and workflow are available to run on single-node machines. But due to the continuous growth in genomics datasets, processing this data on a single node becomes inefficient and time consuming because of Input/Output bottlenecks, limitations on the number of physical cores in a single CPU and memory capacity constraints. To scale up these tools for distributed computing environments, both high performance computing (HPC) programming models (using MPI) and big data frameworks (using Hadoop and Spark) have been explored in the past decade.

MPI (message passing interface) implementations leverage the benefits of distributed memory architectures in inter-node communication. The workflow can exploit the maximum bare-metal performance of such multi-node clusters using shared memory MPI implementations. Previously, too little emphasis has been put on developing MPI based cluster scaled bioinformatics tools and workflows. The reason can be the lack of fault-tolerance [2], redundant data replication, and complexity to develop parallel algorithms in this approach. However, new fault-tolerance models for MPI [3] can enable fault-tolerance mechanisms for such applications and workflows. Similarly, the availability of one-sided communication in new MPI-3 RMA (Remote Memory Access) standard promises better performance gains in the applications while requiring no (or very little) inter-node data sharing and communication. Many tools in a variant calling workflow exhibit such property of not sharing data between the nodes and may run independently (with the exception of sorting).

Apache Hadoop [4] is a MapReduce framework used to process chunks of big datasets in parallel on large cluster nodes in a fault-tolerant and reliable manner. MapReduce usually splits the input data into smaller chunks, runs these chunks completely independently in map tasks, sorts the output of these tasks which is fed to a reduce task as input to generate the final output. MapReduce exclusively uses key-value pair input data to process, sort and aggregate the output based on keys. Hadoop Distributed File System (HDFS) is commonly used to store the input and output data on local compute nodes or on network storage nodes. Some early variant calling workflows like Halvade [5] use this approach to exploit computing cluster resources by running multiple legacy application instances (loosely integrated in the Apache Hadoop Framework) in parallel on chunked input data.

Apache Spark [6] is a unified analytics engine to process big data in a distributed computing environment, with built-in modules for streaming data, distributed machine learning, SQL functions and graph processing. Spark also provides high-level APIs for Java, Scala, Python and R languages. In Spark, resilient distributed datasets (RDDs) are the core components that are distributed across the nodes of a cluster to be operated on in parallel. RDDs can be cached/persisted in-memory across nodes to store intermediate results for iterative processing. Spark commonly uses HDFS to read/write data, but also supports other storage systems like NFS, HBase and Amazon's S3. Many variant calling workflows and tools have been developed over the last decade since its first release, including SparkGA2 [7], ADAM [8], SparkBWA [9], BWASpark [10], PipeBWA [11], etc.

In this article, we propose and implement a new framework that combines the advantage of easy programmability of Apache Spark and the high efficiency of MPI. The resulting framework integrates Apache Spark NGS data pre-processing

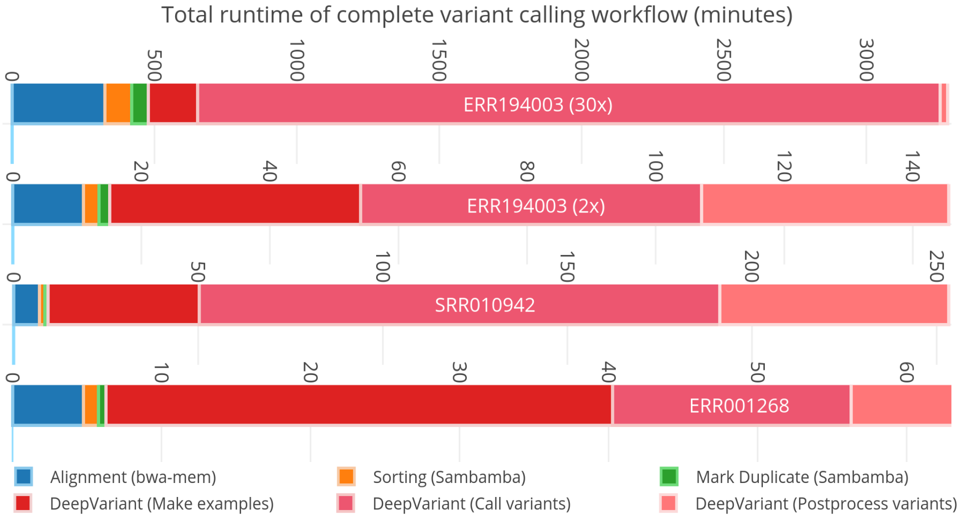


Figure 4.1 Single node total runtimes for complete variant calling workflow using DeepVariant for different datasets.

with the Apache Arrow in-memory data format. Our framework tightly integrates pre-processing (reads sorting and duplicate removal) applications in Python using distributed Dataframes (DF) based sorting and vectorization. This is the first ever such implementation for genomics data to exploit the benefits of Apache Arrow in-memory data format in Apache Spark. The key contributions of our approach are as follows:

- The first scalable approach for DNA data pre-processing that uses Apache Arrow for efficiently utilizing compute resources while preserving easy programmability
- Improved performance of up to 2x compared to state-of-the-art scalability approaches
- Integration with DeepVariant to create the first scalable open source DeepVariant workflow on Apache Spark

This article is organized as follows. In Section "Background and related work", we discuss single node and cluster scaled pre-processing and variant calling workflows, followed by Section "Methods" which presents the in-depth details of the new Apache Arrow based data format for NGS data. In Section "Design and Implementation", the internal design flow and implementation details of our new efficient workflow are discussed. Furthermore, Section "Results and Evaluation" describes the results of our implementation using different nodes configurations with different sequencing coverage/depth datasets to show the scalabil-

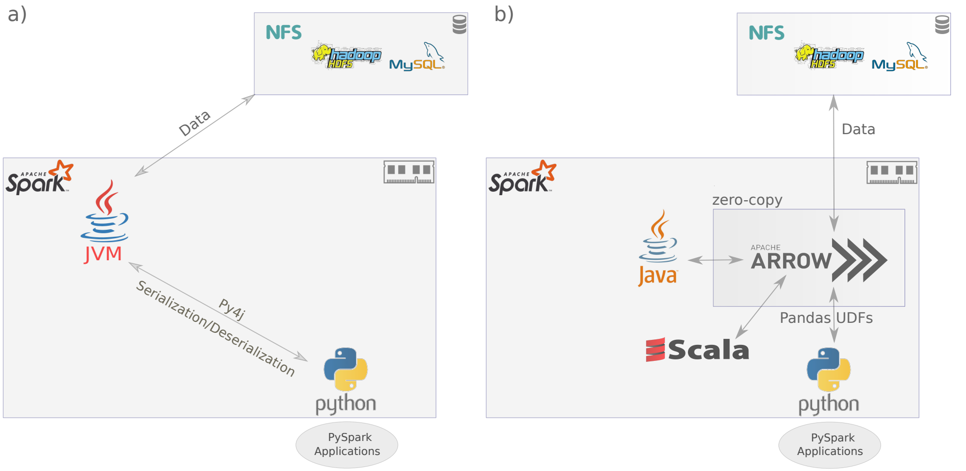


Figure 4.2 **a)** Python programs in Spark require inefficient data serialization/deserialization between Python and JVM processes (using the Py4j library). **b)** Efficient data communication between frameworks/languages using Apache Arrow unified in-memory columnar data format with zero-copy overhead and different languages APIs/interfaces availability in Spark cluster.

ity and the performance comparisons with state-of-the-art methods. In Section "Discussion", more detailed insights on performance, scalability, resources utilization and memory consumption are given. Finally, Section "Conclusion" ends with some concluding remarks and possible future directions.

4.3. Background and related work

In this section, first we introduce and discuss some tools used to pre-process NGS data followed by a discussion of some widely used cluster scale variant calling workflows.

4.3.1. Pre-processing NGS data

Pre-processing of NGS data requires a number of steps: 1. alignment of raw FASTQ data against a reference genome, 2. chromosome based coordinate sorting, and 3. Polymerase Chain Reaction (PCR) duplicates removal (optional, only required if data is not PCR-free or in some datasets for better accuracy). These steps are common in all most every variant calling workflow. There are many tools available publicly to pre-process NGS data efficiently on single node machines. Bowtie2 [12] and BWA-MEM [13] tools are widely used for short read sequence alignments.SAMtools [14], Picard [15], Sambamba [16] and samblaster [17] are some of the most famous and widely used tools for the purpose of indexing, sorting, and duplicates removal in SAM/BAM/CRAM files.

4.3.2. Variant calling

Variant calling reveals deep insights into nucleotide-level organismal differences in some specific traits among populations from an individual genome sequence data. It discerns genetic variations in three categories like single nucleotide polymorphisms (SNPs), insertions and deletions (indels), and/or structural variants (SVs, may also include Copy Number Variations (CNVs), duplication, translocation, etc). The GATK HaplotypeCaller is a widely used variant caller to detect germline variations. DeepVariant [18] is being considered a more accurate germline variants caller for both short and long reads. Tools like VarScan [19], VarDict [20], MuTect2 [21] are used for somatic variant calling analysis. NeuSomatic [22, 23] is deep convolutional neural networks based somatic variant caller which runs in both standalone and ensemble modes (MuTect2, MuSE, Strelka2, SomaticSniper, VarDict, and VarScan2) for accurate somatic variants detection. Octopus [24], FreeBayes [25], Strelka2 [26], SNVer [27] and LoFreq [28] are also used for both germline and somatic variant calling analysis. The DeepVariant variant caller based workflow outperforms in both PrecisionFDA (pFDA) Challenges v1 [29] (highest SNP performance) and v2 [30] (all benchmark regions for PacBio and multi, difficult-to-map regions for ONT). DeepVariant does not require some additional pre-processing steps like base quality recalibration. Therefore we selected this variant caller to integrate with our pre-processing workflow. As shown in Figure 4.1, we run the fastest pre-processing tools with DeepVariant on a single machine with different datasets to get an idea of individual tool runtime in the workflow.

4.3.3. Cluster scaled workflows

There are many cluster scaled multi-node implementations available for alignment using both HPC languages like MPI/Unified Parallel C (UPC) as well as big data framework like Hadoop MapReduce and Apache Spark. pBWA [31] and mpiBLAST [32] use MPI and CUSHAW3 [33] uses UPC++. Similarly ADAM's Cannoli [8], SparkBWA [9] and PipeMEM [11] are a few Apache Spark based BWA-MEM implementations that use BWA-MEM as loosely integrated underneath these implementation while GATK BWASpark modifies the original BWA-MEM to exploit the Spark scheduling and shuffling functionality to run BWA-MEM instances in parallel on clusters.

ADAM, Halvade and SparkGA2 are few implementations that also handle whole variant calling workflows based on GATK best practices including alignment, sorting, duplicates removal and base quality score recalibration.

ADAM, Halvade and SparkGA2 use the built-in Scala API in Spark for sorting the aligned reads. As Picard MarkDuplicate algorithm is considered as standard for paired-end reads for duplicates removal, SparkGA2 and Halvade use this Picard MarkDuplicate tool in Spark for distributed processing on cluster while ADAM has implemented their own duplicates removal algorithm in Scala which is nearly identical to the Picard MarkDuplicate algorithm. A more detailed comparison of these workflows for each individual pre-processing stage output storage strategy is given in Table 4.1.

4.3.4. Apache Arrow in Apache Spark

Apache Arrow [34] is an in-memory standard columnar data format. Apache Arrow also provides API interfaces and functions to process datasets in Go, C, C++, C#, Java, JavaScript, R, Rust, MATLAB, Ruby and Python languages. Due to the columnar data storage, efficient vectorized data analytics operations and better cache locality can be exploited. This in-memory format also supports zero-copy reads for large datasets in inter-process communication without serialization/deserialization overheads. Figure 4.2 shows how a common Apache Arrow based data format is being used in Apache Spark with different language interfaces.

Apache Spark leveraging Apache Arrow [35]: In this paper, we use Python as the language to implement our workflow due to its high level of abstraction and ease of implementation. It also has a stable API to Apache Arrow used in Apache Spark to efficiently transfer data between JVM and Python processes.

Pandas user-defined functions (UDFs): The Python computation model in PySpark on UDFs is scalar, i.e., during UDF evaluation, the JVM executor process sends row data to PySpark workers which invoke UDFs on a row-by-row basis and send the results back to the executor process. However, the current Spark/PySpark release uses immutable Arrow RecordBatches (RBs) data instead of Spark built-in row based data. This enables vectorized UDFs evaluation on these RBs using Pandas Dataframes, which in turn gives a huge performance improvement. Due to vectorized UDF operations, the reduced number of system calls enables faster I/Os.

As traditionally Apache Spark uses a row based memory layout, using Arrow RBs requires converting Spark row based data to Arrow RecordBatch and vice versa to apply vectorized UDF operations in Pandas Dataframes. Some other operations (like grouped data in Pandas Dataframes on UDFs, and converting Spark Dataframes to/from Pandas Dataframes) are also becoming more efficient using Arrow underneath, which is discussed in more details in Section "Methods".

Table 4.1 A comparison of NGS data pre-processing workflows with their output storage approaches for each stage.

Framework	Alignment (output)	Sorting (output)	Duplicates removal (output)
Halvade	*.SAM in disk	in-memory (elPrep)	in-memory (elPrep)
SparkGA2	*.fq.gz in disk	*.BAM in disk	*.BAM in disk
ADAM	ADAM Parquet in disk	ADAM Parquet in memory	ADAM Parquet in memory
VC@Scale (this work)	in memory (Apache Arrow RecordBatches)	in memory (PySpark DFs)	in memory (PySpark DFs -> *.BAM)

Pandas function APIs: Python native functions can be applied on PySpark Dataframes, which input/output Pandas instances. Grouped map, map, cogrouped map are a few Pandas API functions to apply on PySpark Dataframes. These functions use Arrow to transfer data and Pandas to work on that data. These functions share the same characteristics as those of Pandas UDFs.

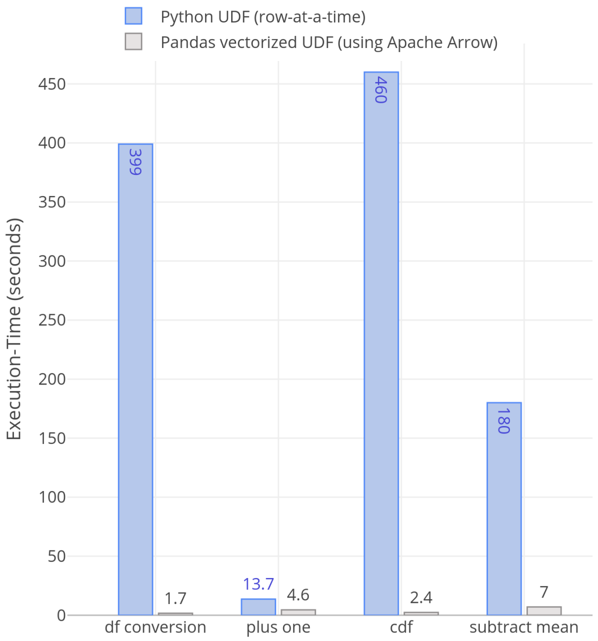


Figure 4.3 Performance comparison of Pandas dataframe to PySpark dataframe conversion using Arrow and without Arrow and Python UDF (row-at-a-time) and Pandas vectorized UDF (using Apache Arrow) operations: plus one, cdf and subtract mean.

UDF performance with/without Arrow: The Spark Python API supports UDFs which operate one-row-at-a-time, resulting in a large serialization and invocation overhead. Apache Arrow based unified memory format brings the benefits of high performance and low overhead dataframes conversion (PySpark<->Pandas) and vectorized Pandas UDFs operations in Python native environments. Because Spark inherently operates on row based memory layouts and Arrow data format is columnar which requires row-column conversions (Spark row <-> Arrow RecordBatch) overhead when doing these operations. In Figure 4.3, we show the performance comparison of 1) converting a Pandas dataframe to PySpark dataframe with Arrow and without Arrow, 2) Python UDF (row-at-a-time) and Pandas vectorized UDF (using Apache Arrow) for plus one, 3) cumulative probability distribution function (cdf), and 4) subtract mean examples [36].

4.4. Methods

In this section, we discuss the details of architectural approaches we have adopted in this work for processing the variant calling workflow.

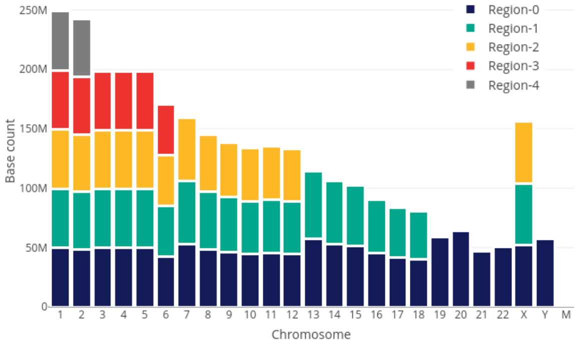


Figure 4.4 Static load balancing technique adopted in this work for BWA-MEM output which divides chromosomes based regions to join and process them in parallel for all further workflow stages.

Overview

The benefits of using distributed big data frameworks to process genomics data are fourfold: they provide easy and flexible deployment, efficient cluster scalability, fault-tolerance, as well as cheaper costs on public clouds and private HPC clusters. Traditionally, these frameworks use distributed file systems like the Hadoop distributed file system (HDFS) or the Network File System (NFS) for storage. The intermediate processing stages place data in-memory on-demand if enough memory is available in the form of RDDs. RDDs generally store data in their internal row format while Apache Arrow provides an efficient columnar data format to create distributed RDDs of Arrow RecordBatches object types.

To validate the scalability and performance advantage of our Apache Arrow based in-memory data placement, shuffling, conversion and computation techniques in Apache Spark using PySpark, we present the design methods for a full variant calling workflow. We have also developed high performance and scalable but very simple, portable and stand-alone methods for BWA-MEM and DeepVariant scalability on HPC clusters using traditional I/O based storage.

Variant calling workflow

In this subsection, we describe the various stages of the variant calling workflow that we designed, as shown in Figure 4.5. We start with the implementation of the pre-processing stages (alignment, sorting and duplicate removal) using Apache Arrow in-memory data format for temporary data storage in Plasma Stores, shuffling/conversion of data and transformations/computations on this data. The resultant data from these pre-processing stages is saved in BAM format. Each BAM file contains the reads of a particular chromosome and a specific region inside a chromosome. Variant caller (DeepVariant) instances process these BAM files on worker nodes and produce VCF files which are merged to produce a final VCF file.

FASTQ chunks streaming

We use the SeqKit [37] to create the FASTQ input chunks in parallel with BWA-MEM for input paired-end NGS data as shown in ❶ of Figure 4.5. SeqKit is an efficient multi-threaded utility, through which we provide this FASTQ data to BWA-MEM instances in streaming fashion, without the need to create FASTQ chunks separately. The number of created FASTQ chunks can be configured in the SeqKit command option, depending on the number of nodes available in the Spark cluster.

Arrow integration in BWA-MEM

BWA-MEM is the most popular alignment tool in the bioinformatics community due to its efficient and accurate alignment algorithm for short reads. In our implementation, each Spark cluster worker node runs one BWA-MEM instance as shown in ❷ of Figure 4.5. We have modified BWA-MEM to output in-memory key-value pair SAM data instead of creating tab-delimited SAM files.

Key-value pairs: Key-value pair based data has proven efficient sorting performance as compared to text/columnar data structures. For every read, after creating its respective SAM fields we convert the whole read SAM data into a key-value pair `<POS:SAM>` and with `RNAME`, an extra information in the structure to store it in a designated immutable Arrow RecordBatch. Each RecordBatch is a combination of a schema, which specifies the types of data fields and the data itself. In our case, `POS` field is integer (Int) type while `SAM` and `RNAME` fields are String type.

Static load balancing: Due to the size differences in the chromosomes of the human genome, we created chromosomes regions for efficient scalability in BWA-MEM and the same such trend is followed in subsequent pre-processing stages as well. The number of regions is different for each chromosome to store reads corresponding to their respective regions as shown in Figure 4.4. Each region in each chromosome is on average equal to 40-50 million bps.

Plasma Object Store: The Plasma Object Store is an inter-process communication (IPC) component of Apache Arrow that handles shared memory pools across different heterogeneous systems [38]. To perform IPC, processes can create Plasma objects inside the shared memory pool that are typically data buffers underlying an Arrow RecordBatch. We cannot use more than half of overall system memory for these Plasma Stores. Through the shared memory pool, Plasma enables zero-copy data sharing between the processes. The output SAM data from BWA-MEM instances on each node is being stored in key-value pairs in respective chromosomal regions using the Arrow in-memory format as shown in ❸ of Figure 4.5.

flatMap() on BWA-MEM: We apply the PySpark `flatMap()` function on BWA-MEM instances which use an already SparkContext parallelized/distributed collection of input FASTQ chunks described in Section FASTQ chunks streaming. All the BWA-MEM instances create Arrow RecordBatches of regions individual chromosomes on their own respective nodes. These Batches are temporarily placed

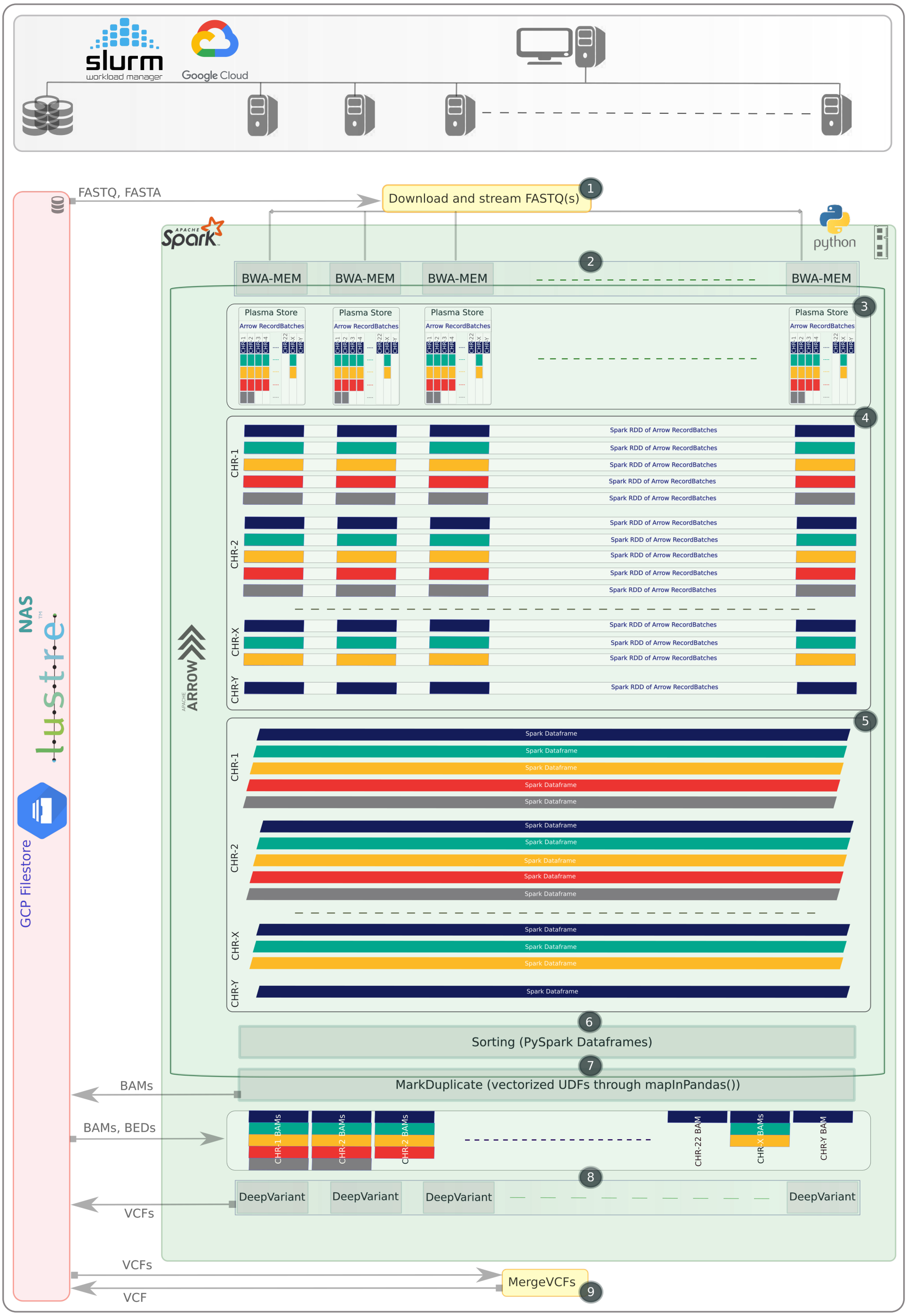


Figure 4.5 Complete design flow of the variant calling workflow implementation in VC@Scale, this design encompasses Slurm Spark/GCP DataProc cluster, Lustre/GCP Filestore as file system, Apache Arrow as in-memory data format for pre-processing and DeepVariant as variant caller.

in Plasma Object Stores on each node.

RDDs of Arrow RecordBatches: As soon as the alignment process on Apache Spark worker nodes finishes, we create distributed RDDs of these Arrow RecordBatch objects available across all the Spark worker nodes as shown in ❹ of Figure 4.5. Each RDD occupies the RecordBatches of a particular chromosome (with its specific region) distributed among all the worker nodes. Arrow RecordBatches are filtered out in this step and cached into the Spark context of the master node.

RDDs to Dataframe: These RecordBatches in RDDs are serialized and a PySpark schema is generated through corresponding Arrow schema enclosed in these RecordBatches. Python objects to Java object conversion on RDDs is then applied as shown in ❺ of Figure 4.5. Finally, these resultant RDDs are converted to Spark Dataframe through Scala `PythonSQLUtils` methods. At this point, we have distributed Spark Dataframes of specific regions of each chromosome. We process these specific chromosomes regions independently and in parallel in the next sorting and duplicate removal stages.

Sorting

All the Spark Dataframes containing specific chromosome regions are sorted (❻ Figure 4.5) by coordinates through `df[n].orderBy('beginPos', ascending=True)` function. This function is very fast and efficient in sorting huge distributed Dataframes. All the Dataframes are sorted in parallel using the Python multiprocessing library `Pool` method.

Duplicates removal

Duplicate removal algorithms in this implementation were written from scratch in Python for both single and paired-end reads. These algorithms are developed using Pandas UDFs to apply on PySpark Dataframes which can use the Pandas function APIs (`df[n].groupby().applyInPandas()`) to leverage the benefits of Arrow for data transfer/conversion and transformations (❼ Figure 4.5). For paired-end reads, a Picard MarkDuplicate compatible algorithm has been developed. The accuracy of this algorithm is validated using different datasets, so that they can be used as a cluster scalable replacement for the existing Picard MarkDuplicate algorithm.

DeepVariant integration

DeepVariant is a being considered as an accurate variant caller for detection of both SNPs and indels variants in germline datasets. Their published results show that DeeVariant performs best for most PrecisionFDA Truth Challenge datasets [39]. We have observed that on a single node, DeepVariant scales very well up to 6-12 threads. Therefore we have enabled running multiple DeepVariant instances on each Spark worker node using the PySpark `flatMap()` function (❽ Figure 4.5). Each of DeepVariant instance takes input BAM (and BED as well in case of WES data) and reference FASTA from the I/O based NFS and produces individual VCF/gVCF files.

VCFs merge

Finally, the individual VCFs created through DeepVariant instances are merged (❾ Figure 4.5) through Samtools to produce a final complete VCF file(s) for further downstream analysis.

Standalone implementations

In addition to implementing the complete workflow, we can also use BWA-MEM and DeepVariant as scalable stand-alone implementations capable of scaling almost linearly on HPC clusters depending on the input data size and number of nodes available.

BWA-MEM: Almost all BWA-MEM cluster scaled implementations (SparkBWA [9], BWASpark [10], PipeMEM [11], ADAM [8], and SparkGA2 [7]) run multiple BWA-MEM instances on each Spark worker node as Spark tasks, which degrades the underlying efficient single node multi-threaded scalability of this tool. Instead we use one BWA-MEM instance on each Spark worker node, storing output SAM files on storage and merging these SAM files to generate a single output SAM file.

DeepVariant: We use Samtools to generate different BAM files representing chromosome regions from a single BAM file in accordance with our human chromosome regions based approach as discussed earlier in Section Arrow integration in BWA-MEM. Similarly, we have divided the reference FASTA into individual chromosome based FASTA files using faSplit [40]. The VCF/gVCF output files of these instances can be merged through Mergevcf or Samtools.

4.5. Results and evaluation

In this section, first we shortly describe the datasets and HPC infrastructure used in the evaluation of our techniques. In addition, we compare our results with other state-of-the-art frameworks for both pre-processing and variant calling stages followed by a detailed analysis and comparison of scalability, performance and speedups with these frameworks.

Datasets

We use multiple whole genome sequencing datasets with varying coverage depth to analyze the maximum possible scalability and performance of our methods. The first dataset is sample ERR001268 from the 1000 Genomes Project (phase 3) Illumina HiSeq generated WGS paired-end read data of NA12878 [41]. In addition, we used Illumina HiSeq 2000 paired-end NA12878 cell line data sequencing sample ERR194003 [42] with sequencing coverage of 30x. We also used 300x sequencing coverage WGS data from Genome in a Bottle (GIAB) aligned with novoalign for the Illumina HiSeq 300x reads for NA12878 [43] to analyze the scalability of DeepVariant. Human Genome Reference, Build 37 (GRCh37/hg19) [44] is used as a reference genome.

Evaluation HPC cluster

All experiments and comparisons are performed on the SurfSara Cartesius [45] HPC cluster (part of the Dutch national supercomputing infrastructure). Each CPU-only node is equipped with a dual socket Intel Xeon Processor (E5-2695 v2 or E5-2690 v3) running at 2.4/2.6GHz. Each processor has 12 physical cores with support of 24 hyper-threading jobs. Similarly, each CPU+GPU node is equipped with a dual socket Intel Xeon Processor (E5-2450 v2) running at 2.5GHz and 2x NVIDIA Tesla K40m GPGPUs. Each processor has 8 physical cores with support of 16 hyper-threading jobs. A total of 64-GBytes (E5-2695 v2/E5-2690 v3) and 96-GBytes (E5-2450 v2) of DDR4 DRAM with a maximum of 59.7 GB/s bandwidth is available for the whole system. A local storage of 1-TBytes and the same amount of network attached storage is available on the system. All nodes are connected through Mellanox ConnectX-3 or Connect-IB InfiniBand adapter.

Lustre [46] distributed and parallel file system is attached to our evaluation HPC cluster. Lustre file system has similar performance as of HDFS/YARN-based Hadoop cluster for shuffle-heavy workloads in Apache Spark.

Red Hat Enterprise Linux operating system is installed on all nodes. Apache Spark cluster is created in deploy-mode ’client’ thorough Slurm [47] Workload Manager and all workflows are executed through bash scripts.

We also used a Google GCP DataProc cluster and Google cloud Filestore, a network attached storage (NAS) to reproduce and run this approach on public cloud environments. All the required applications are installed on Dataproc custom image which is based on the DataProc 2.0.1-ubuntu18 operating system. A detailed description and quick start guide to run all methods in this approach are given on the project github page.

Pre-processing (BWA-MEM, Sorting, Duplicates removal)

Our approach performs pre-processing in a more tightly coupled fashion (i.e., using native PySpark functions) as compared to alternative solutions such as SparkGA2 which stores the output of each of the pre-processing stages to storage and loads it again for subsequent stages. We have tested the scalability and performance of our architectural choices with that of SparkGA2 and ADAM for different cluster sizes; 2, 4, 8 and 16 nodes have been used in almost all comparisons. Storing BWA-MEM output to in-memory key-value pairs using the Arrow format involves almost zero cost overhead for loading data to the next sorting stage. The only data transformation that happens between the alignment and sorting stages is the conversion of RDDs containing Arrow RecordBatch objects to PySpark Dataframes. This transformation is handled through the Apache Arrow APIs internally. A similar key-value pairs transformation of sorted Dataframes to SAM values occurs before the MarkDuplicate stage. Compared to SparkGA2

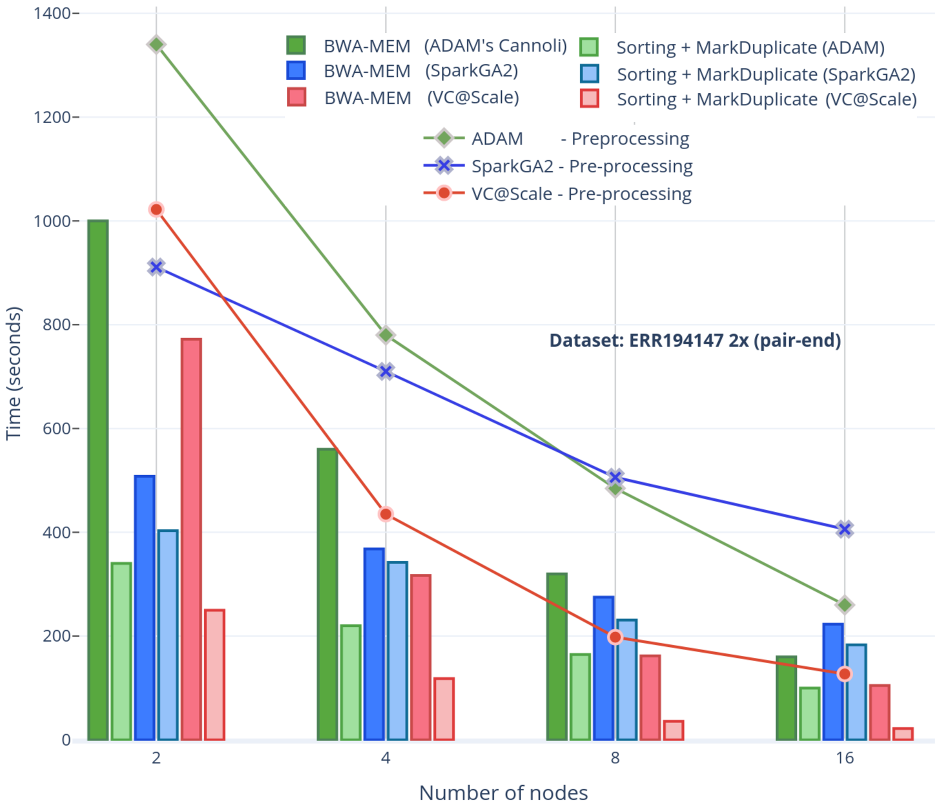


Figure 4.6 VC@Scale, SparkGA2 and ADAM comparisons of scalability for pre-processing stages using different number of nodes for ERR194003 (2x) dataset.

and ADAM pre-processing results, more than 2x speedup is achieved for all cluster sizes and for both ERR001268, and ERR194003 (2x) datasets for SparkGA2 while 2-4x speedup is achieved as compared to ADAM workflow pre-processing, as shown Figure 4.6 and Figure 4.7, respectively.

Variant calling (DeepVariant)

DeepVariant is about 3x to 4x slower than GATK's HaplotypeCaller on CPU-only machines [48]. To make it scalable for clusters, we run each chromosome region independently on a different Spark worker node. In our pre-processing stage, we already store the load-balanced BAMs as individual chromosome regions. This approach provides a very fruitful base for a subsequent variant calling stage (DeepVariant in our case). For DeepVariant CPU-only version, we used a CPU cluster with different number of nodes (2, 4, 8, 16, and 32) and with multiple datasets like ERR001268, ERR194003 (2x), ERR194003 (30x) and NA12878 (300x). In Fig-

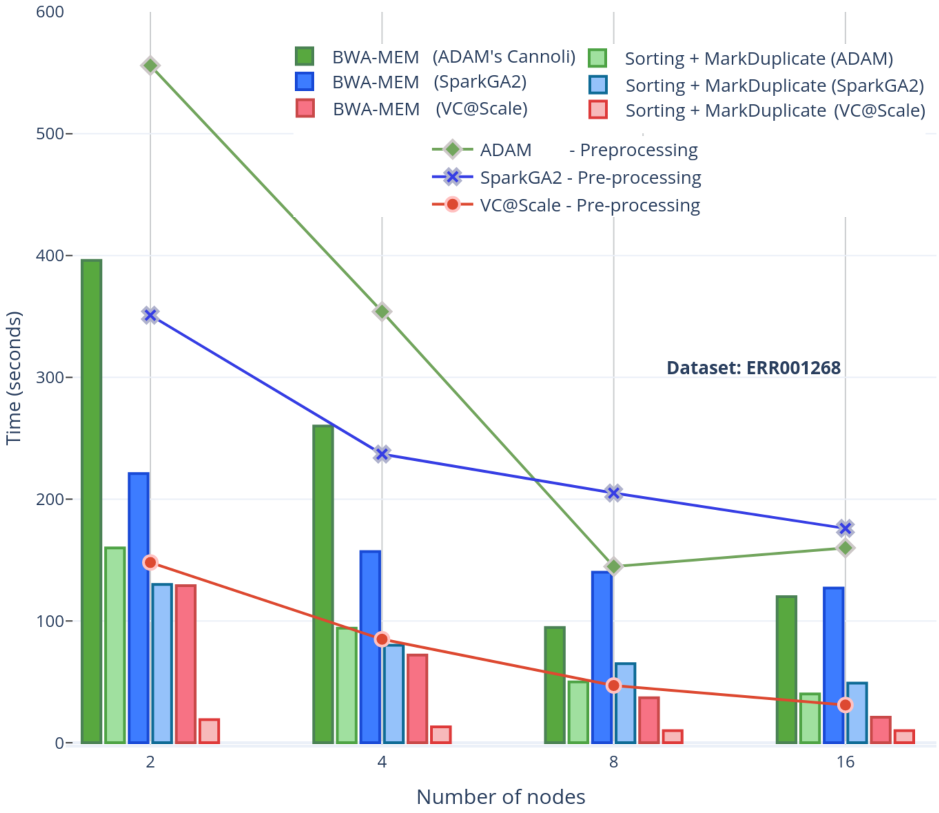


Figure 4.7 VC@Scale, SparkGA2 and ADAM comparisons of scalability for pre-processing stages using different number of nodes for ERR001268 dataset.

ure 4.10, the results show an increasing speedup for DeepVariant scalability on a Spark cluster. In DeepVariant some smaller datasets perform best with just 16 nodes, while the processing trend of other datasets show even more scalability when we increase the nodes from 16 to 32. The total runtime is decreased up to 8x as compared to a single CPU machine. DeepVariant consist of three steps: 1) make_examples, 2) call_variants and 3) postprocess_variants. The first two steps are the most time consuming (see Figure 4.1). To improve their performance, the make_examples step is multi-threaded for reading inputs and creating examples, while call_variants has been accelerated for GPUs. As shown in Figure 4.8, we have observed in some datasets like ERR194003 (30x) that the call_variants step takes up to 95% of the total time of DeepVariant. This step can be accelerated on GPUs with almost 10x as shown in the GPU accelerated results of Figure 4.8. Such acceleration makes DeepVariant more feasible to be adopted in practice. We also use a GPU cluster to test our approach for DeepVariant scalability as well as acceleration. Results in Figure 4.11 show more than 2x speedup with

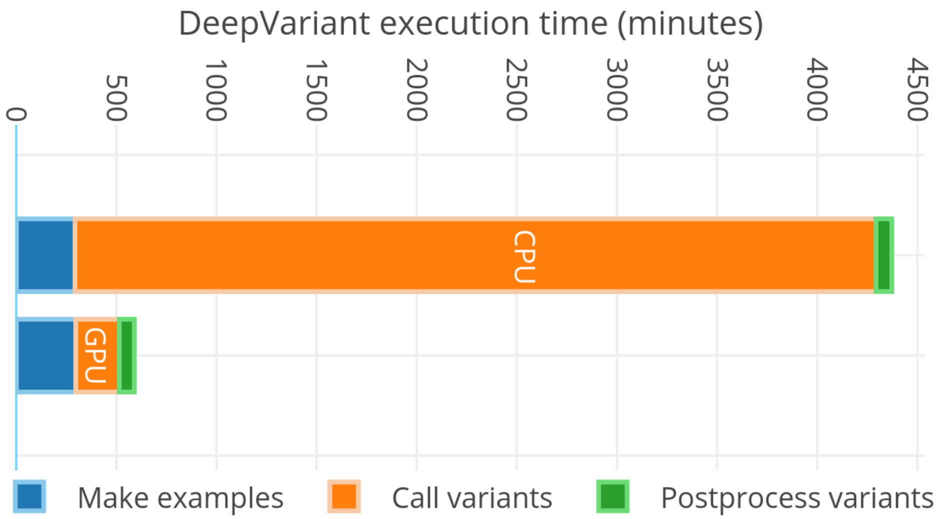


Figure 4.8 Single node CPU-only and GPU accelerated DeepVariant for ERR194003 (30x) dataset.

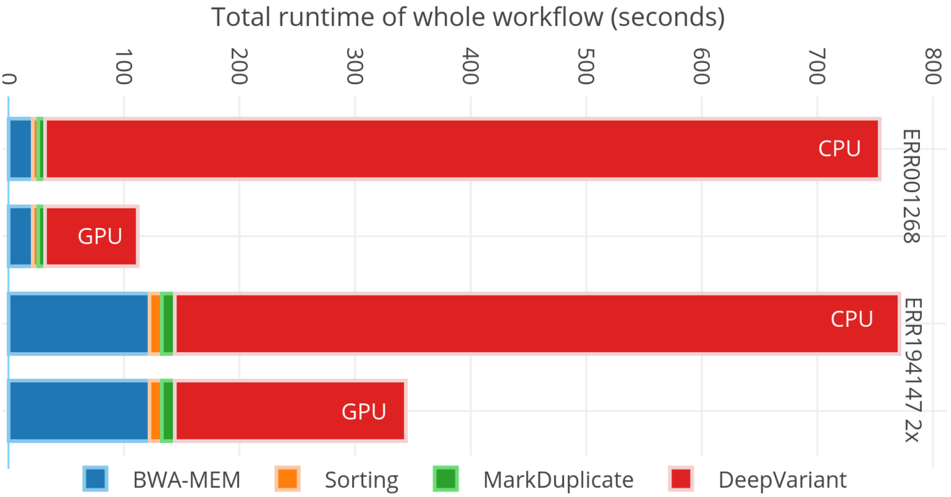


Figure 4.9 Total runtime for DeepVariant based complete variant calling workflow (VC@Scale) which uses best performance combination of nodes. For both datasets preprocessing (BWA-MEM, Sorting and MarkDuplicate) is using 16 nodes while 32 nodes are used for DeepVariant.

GPU accelerated DeepVariant for the ERR194003 (30x) dataset as compared to CPU-only.

Variant calling workflow

The total runtime results for whole variant calling workflow using BWA-MEM, Sorting, MarkDuplicate and DeepVariant are shown in the Figure 4.9. Here we show the best possible nodes configuration for both, pre-processing and variant calling stages. For the dataset ERR194003 (2x), in pre-processing 16 nodes are the best fit while 32 nodes give better scalability in variant calling. Similarly for dataset ERR001268, 16 nodes provides best performance and scalability. The total runtime is decreased by up to 5x as compared to a single CPU machine.

Standalone BWA-MEM & DeepVariant

Our workflow can also be used as two independent components: a standalone BWA-MEM and a standalone DeepVariant component. The BWA-MEM component represents the fastest standalone Spark-based scalable implementation compared to other state-of-the-art BWA-MEM cluster solutions. In this solution we achieve almost linear speedups with increasing the number of nodes. The output is saved into separate SAM files which can be merged through Samtools to output a single SAM file.

In this solution, an already created BAM file can be used with DeepVariant for variant calling on cluster. As discussed earlier in Section Standalone implementations, we used Samtools to split the BAM file into our pre-defined chromosome regions to generate load-balanced chromosome regions parts. In this way we ran DeepVariant instances on Spark worker nodes. The output speedup and scalability results are the same as mentioned in Section Variant calling (DeepVariant).

Standalone pre-processing (Piped)

In a standalone pre-processing pipeline, we use the following tools: BWA-MEM, Sambamba (sorting, markdup) & Samtools (merge). This is a simple and efficient implementation of pre-processing stages (alignment, sorting and markduplicate) on a Spark cluster. We integrated already existing and widely used tools in this workflow. Sambamba sorting and MarkDuplicate algorithms produce the same output as Picard's. In this approach, the master node streams the FASTQ data to all worker nodes as discussed in Section FASTQ chunks streaming. All worker nodes initiate one BWA-MEM instance. The BWA-MEM output is then piped into Sambamba which performs both SAM to BAM conversion and sorting. The Sambamba MarkDuplicate stage is optional. After these stages, we use the Samtools merge algorithm to combine all the resultant BAM files into a single BAM file. We have developed a demo with different nodes on a Google GCP DataProc cluster, which is publicly available and can be tested with GCP. A complete guide to execute this workflow is available on our project github page [49].

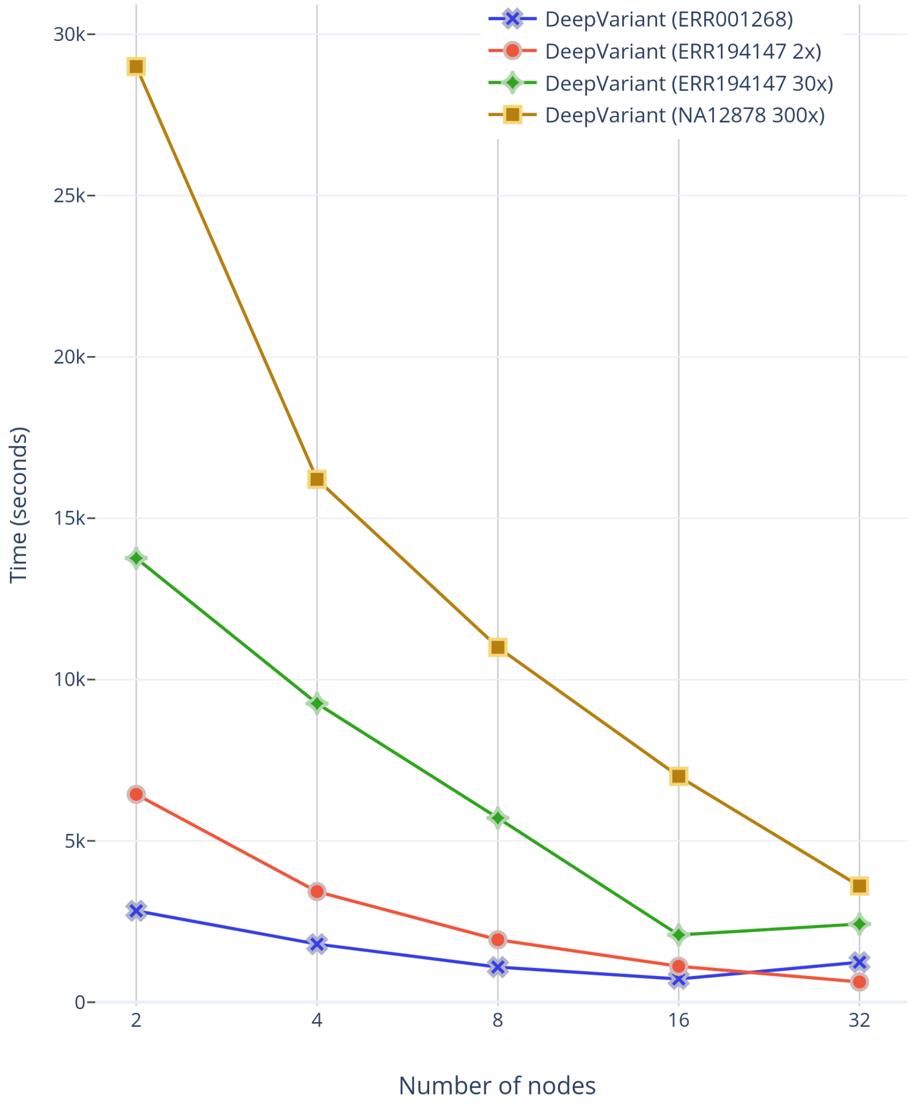


Figure 4.10 VC@Scale-DeepVariant scalability for different datasets and the number of nodes used in each run.

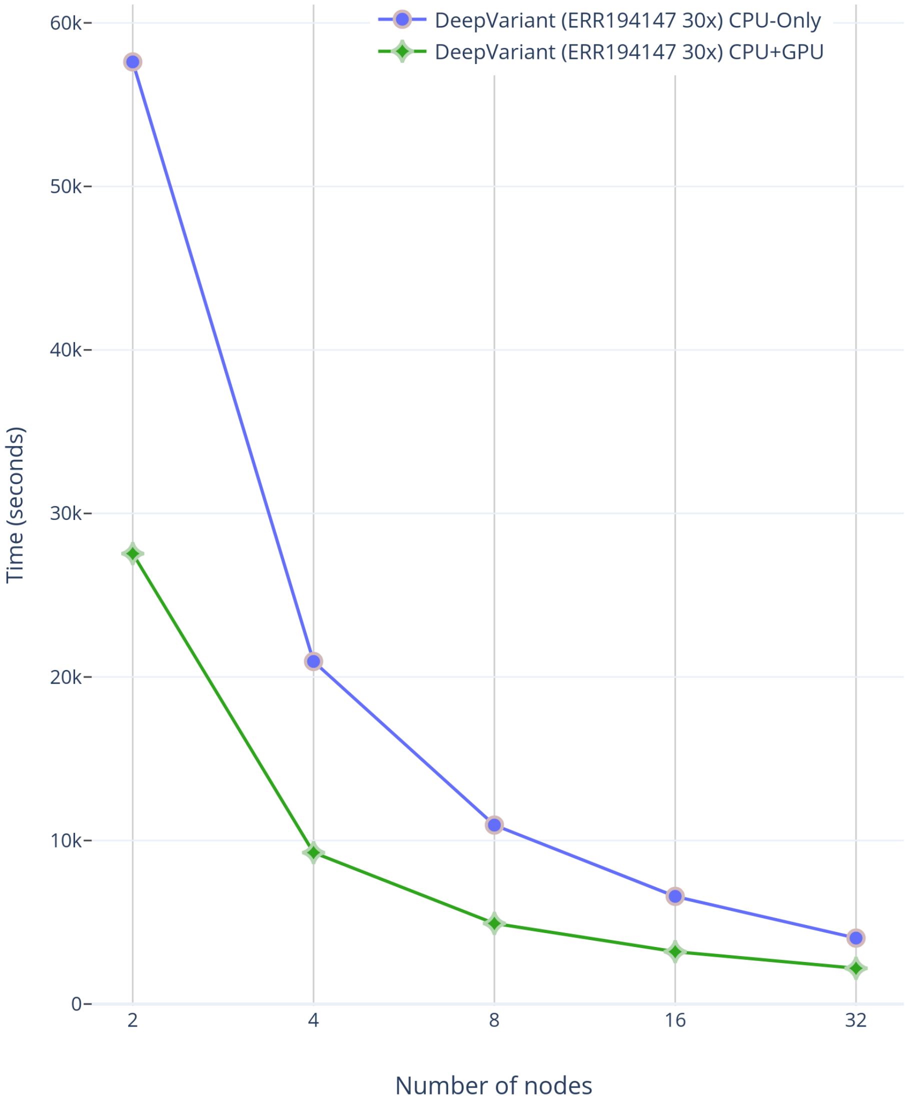


Figure 4.11 GPUs accelerated VC@Scale-DeepVariant scalability for ERR194003 (30x) dataset.

Other variant callers support/integration

Any variant caller which can support region-specific variant calling can be integrated into this workflow. We integrate Octopus [24], a recent and accurate/fast variant caller as a use case to demonstrate the feasibility of integrating other variant callers in this approach. We also performed a comparison on DeepVariant and Octopus on Chr20 - HG003 Illumina WGS reads publicly available from the PrecisionFDA Truth v2 Challenge and we found Octopus accuracy is almost identical to DeepVariant for both SNP and INDEL variants. We also provide a guide to reproduce these both use cases on github.

4.6. Discussion

Here we discuss some of the advantages and limitations of our approach, in addition to the advantages of using Apache Arrow as a common in-memory data format for variant calling workflows.

Portability of the implementation

The workflow implementations discussed in this paper are portable to many HPC cluster environments. We use standard cluster solutions such as the Singularity container, and the Slurm Workload Manager to deploy and reproduce them with ease on other cluster environments.

Accuracy

To compare the small variants detection accuracy both in single node (default) method and VC@Scale (distributed method), we used HG002 (NA24385 sample with 50x coverage taken from PrecisionFDA challenge V2) dataset to detect SNP and INDEL variants using DeepVariant (v1.1.0), against GIAB v4.2 benchmark set for HG002 dataset. The GA4GH small variant benchmarking tool hap.py [50] has been used to compare the resulting variants in both methods. Table 4.2 and Table 4.3 list the accuracy analysis results in terms of recall, precision and F1-score. The tables show that in general VC@Scale has very comparable accuracy results to the baseline. Detailed inspection of the results shows that VC@Scale detects the same number of INDEL true positives and false negatives, and a slightly lower number of false positives compared to the baseline. This gives the same recall results, but ensures a slightly improved precision and F1-score. For SNPs, however, VC@Scale detects slightly less true positives but more false negatives and false positives. This gives a marginally degraded SNP recall, precision and F1-score.

Parallelization and scalability

Due to dividing chromosomes based on regions for load-balancing in the alignment stage, better parallelization is achieved per node in both pre-processing and variant calling stages. In the examples in this paper, we created a total of 65 such regions, which allows us to scale up to 32 nodes for the pre-processing

Table 4.2 Accuracy evaluation of small variants of HG002 (NA24385 with 50x coverage taken from PrecisionFDA challenge V2 datasets) against GIAB HG002 v4.2 benchmarking set. This table shows the SNP and INDEL results for "Chr1" on a single node (default) run.

Variant	Truth total	TP	FN	FP	Recall	Precision	F1-Score
INDEL	42689	42390	299	131	0.992996	0.997053	0.995020
SNP	264143	262367	1776	351	0.993276	0.998665	0.995963

Table 4.3 Accuracy evaluation of small variants of HG002 (NA24385 with 50x coverage taken from PrecisionFDA challenge V2 datasets) against GIAB HG002 v4.2 benchmarking set. This table shows the SNP and INDEL results for "Chr1" on a cluster scaled (distributed) VC@Scale implementation. "Chr1" has been chunked into ten parts.

Variant	Truth total	TP	FN	FP	Recall	Precision	F1-Score
INDEL	42689	42390	299	127	0.992996	0.997142	0.995065
SNP	264143	262365	1778	355	0.993269	0.998649	0.995952

and DeepVariant stages. When using 32 nodes, two regions are being mapped to each worker node. The total runtime of the workflow is determined by the slowest node in the cluster. As the size of the input data increases, making smaller regions can give more scalability for higher number of nodes.

Two points are important to understand the scalability and performance predictability of such applications when using the Apache Spark framework. 1. Spark always takes some fraction of time to initialize the underlying processes on its worker nodes and also spends a similar amount of time in scheduling and collecting the result. Therefore, increasing the number of nodes Spark uses, also increases this overhead time. If increasing the number of nodes results in a small overall processing time then it reaches a point where the above mentioned overhead time surpasses the processing time. 2. Data size also influences the scalability and performance of these applications and this is directly linked to our previous point. When we increase the number of nodes, the data size is always divided by the number of nodes being used. So we have to figure out the best possible scenario of performance on the cluster when choosing the number of nodes and data size being used.

System resources utilization

Existing Spark based variant calling workflows like ADAM, SparkGA2 and Halvade launch multiple instances of BWA-MEM on each Spark worker nodes which degrades the actual performance of BWA-MEM instances on each individual node. These workflows store the output of each stage to the disk which incurs sometimes I/O wait overheads as well as reading and writing to I/Os for each stage and parsing text SAM or compressed BAM also involves some additional overheads as shown in Figure 4.12. The figure uses the ERR194003 (2x) dataset with 16 nodes

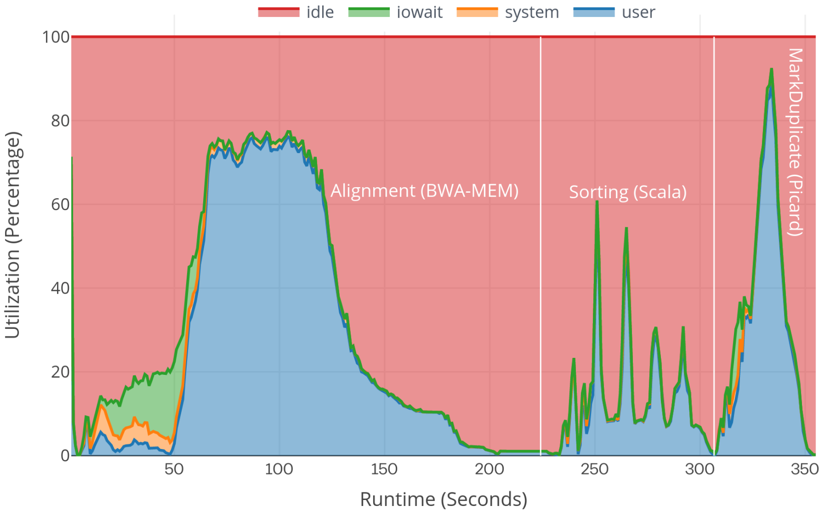


Figure 4.12 SparkGA2 cluster wide system resources utilization graph for pre-processing stages.

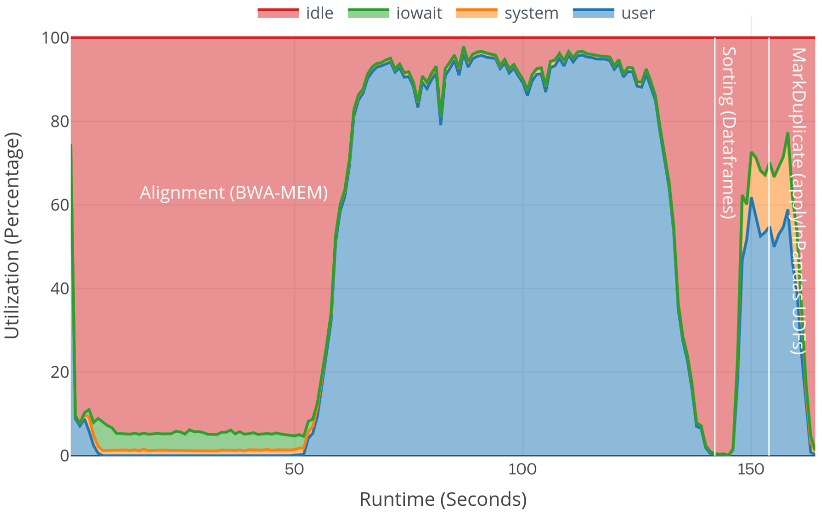


Figure 4.13 VC@Scale cluster wide system resources utilization graph for pre-processing stages.

cluster (the best scalable and optimized use case for both SparkGA2 and in our approach). For comparison, we also show the system resource utilization for our approach in Figure 4.13. In both approaches, the first 50 seconds are spent to load the FASTA index and to read the first FASTQ data chunk. In SparkGA2, the I/O wait time is a bit higher than ours because it loads multiple indices for multiple BWA-MEM instances on each node while we just load one FASTA index on each node. After loading the files, the actual alignment process starts. The figures show that in SparkGA2, a maximum of 78% CPU resources are being used for BWA-MEM while in our approach almost 95% on average CPU resources are being used for BWA-MEM. Similarly, in Sorting only about 10% and in MarkDuplicate 50% on average CPU resources are being used in SparkGA2. In our approach, timing graph shows almost both stages take half of the total time with an average of 60-65% utilization. Because Spark uses lazy evaluations of Dataframes operations, we cannot distinguish exactly the timing for each operation separately. Due to some internal shuffling and the PySpark to Pandas Dataframes conversion via Apache Arrow, a slightly bigger amount of system time is being spent there.

Memory consumption

We use Plasma Object Store to place temporary BWA-MEM output data in-memory on each node. These objects are removed when the Spark Dataframes creation is accomplished. During this intermediate step we use a memory space that is 2x the size of the SAM file. Similarly, during the sorting process, Spark does a lot of internal shuffling which requires additional memory. In MarkDuplicate, we use Pandas UDFs which internally use the Arrow data format for PySpark Dataframes to Pandas Dataframes conversion and vice versa. This step is also memory intensive. This workflow in pre-processing stages, requires 2x memory size as compared to SAM data produced by BWA-MEM stage on that worker node while the master node requires memory size equal to total size of the SAM data produced by all worker nodes. For DeepVariant stage, it only requires a couple of GBytes memory on both worker and master nodes.

4.7. Conclusion

A scalable and high performance DeepVariant based variant calling workflow for cluster scaled environments is presented in this paper. We employ FASTQ data streaming technique to feed data to an alignment stage followed by an in-memory data load-balancing method to store alignment output. Sorting and mark duplicate stages are implemented in such a way to get benefits from the Apache Arrow data format. The load-balanced BAM files output of the pre-processing stages is used in DeepVariant, making variant calling more efficient on a compute cluster.

Scalability analysis of our approach shows significant reduction in runtime compared to a single node. For pre-processing stages, ERR001268 and ERR194003 (2x) datasets provide up to 7x and 8x for 16 nodes, respectively. For DeepVariant, ERR001268 (1x coverage) gives 5x, ERR194003 (2x) gives nearly 8x, ERR194003 (30x) and NA12878 (300x) gives 12x speedup for 32 nodes as compared to single node runtime. Similarly, our approach is faster than state-of-the-art workflows, such as SparkGA2, resulting in 1.8x and 2x speedup for ERR001268 (1x) and ERR194003 (2x) for pre-processing stages on 16 nodes, respectively. Our architectural approache also increase efficient system resource utilization. For pre-processing stages, we achieve 20% to 25% better processor utilization which in turn helps to speedup overall processing. The variants accuracy analysis on PrecisionFDA V2 challenge datasets against the GIAB truth v4.2 benchmark truth data shows almost identical results as compared to single node runs. We also show the flexibility of this approach to adopt other variant callers. We integrate the Octopus variant caller as a use case for this purpose. We also demonstrate the deployment of this approach on public clouds, currently, Google GCP DataProc cluster has been used for this purpose.

REFERENCES

[1] T. Ahmad, Z. Al Ars, and H. P. Hofstee. "VC@Scale: Scalable and high-performance variant calling on cluster environments". In: *GigaScience* 10.9 (Sept. 2021). giab057. issn: 2047-217X. doi: 10.1093/gigascience/giab057. eprint: https://academic.oup.com/gigascience/article-pdf/10/9/giab057/40327053/giab057.pdf. url: https://doi.org/10.1093/gigascience/giab057.

[2] W. Gropp and E. Lusk. "Fault Tolerance in Message Passing Interface Programs". In: *The International Journal of High Performance Computing Applications* 18.3 (2004), pp. 363–372. doi: 10.1177/1094342004046045. eprint: https://doi.org/10.1177/1094342004046045. url: https://doi.org/10.1177/1094342004046045.

[3] F. Cappello, G. Al, W. Gropp, S. Kale, B. Kramer, and M. Snir. "Toward Exascale Resilience: 2014 Update". In: *Supercomput. Front. Innov.: Int. J.* 1.1 (Apr. 2014), pp. 5–28. issn: 2409-6008. doi: 10.14529/jsfi140101. url: https://doi.org/10.14529/jsfi140101.

[4] Apache. *Apache Hadoop [Accessed: 2nd April 2019]*. 2019. url: https://hadoop.apache.org/.

[5] D. Decap, J. Reumers, C. Herzeel, P. Costanza, and J. Fostier. "Halvade: scalable sequence analysis with MapReduce". eng. In: *Bioinformatics (Oxford, England)* 31.15 (Aug. 2015). btv179[PII], pp. 2482–2488. issn: 1367-4811. doi: 10.1093/bioinformatics/btv179. url: https://doi.org/10.1093/bioinformatics/btv179.

[6] Apache. *Apache Spark: Lightning-fast unified analytics engine [Accessed: 2nd April 2019]*. 2019. url: https://spark.apache.org/.

[7] H. Mushtaq, F. Liu, C. Costa, G. Liu, P. Hofstee, and Z. Al-Ars. "SparkGA: A Spark Framework for Cost Effective, Fast and Accurate DNA Analysis at Scale". In: *Proceedings of the 8th ACM International Conference on Bioinformatics, Computational Biology,and Health Informatics*. ACM-BCB '17. Boston, Massachusetts, USA: ACM, 2017, pp. 148–157. isbn: 978-1-4503-4722-8. doi: 10.1145/3107411.3107438. url: http://doi.acm.org/10.1145/3107411.3107438.

[8] M. Massie, F. Nothaft, C. Hartl, C. Kozanitis, A. Schumacher, A. D. Joseph, and D. A. Patterson. *ADAM: Genomics Formats and Processing Patterns for Cloud Scale Computing*. Tech. rep. UCB/EECS-2013-207, EECS Department, University of California, Berkeley, 2013.

[9] J. M. Abuín, J. C. Pichel, T. F. Pena, and J. Amigo. “SparkBWA: Speeding Up the Alignment of High-Throughput DNA Sequencing Data”. In: *PLOS ONE* 11.5 (May 2016), pp. 1–21. doi: 10.1371/journal.pone.0155461. url: https://doi.org/10.1371/journal.pone.0155461.

[10] B. Institute. *BWA on Spark*. 2018. url: https://gatk.broadinstitute.org/hc/en-us/articles/360037225092-BwaSpark-BETA-.

[11] L. Zhang, C. Liu, and S. Dong. “PipeMEM: A Framework to Speed Up BWA-MEM in Spark with Low Overhead”. In: *Genes* 10.11 (Nov. 2019), p. 886. issn: 2073-4425. doi: 10.3390/genes10110886. url: http://dx.doi.org/10.3390/genes10110886.

[12] B. Langmead and S. L. Salzberg. “Fast gapped-read alignment with Bowtie 2”. In: *Nature Methods* 9.4 (Apr. 2012), pp. 357–359. issn: 1548-7105. doi: 10.1038/nmeth.1923. url: https://doi.org/10.1038/nmeth.1923.

[13] H. Li and R. Durbin. “Fast and accurate short read alignment with Burrows–Wheeler transform”. In: *Bioinformatics* 25.14 (May 2009), pp. 1754–1760. issn: 1367-4803. doi: 10.1093/bioinformatics/btp324. url: https://doi.org/10.1093/bioinformatics/btp324.

[14] H. Li. “The Sequence Alignment/Map format and SAMtools”. In: *Bioinformatics* 25 (Jan. 2009), pp. 2078–2079.

[15] *Picard toolkit*. http://broadinstitute.github.io/picard/. 2019.

[16] A. Tarasov, A. J. Vilella, E. Cuppen, I. J. Nijman, and P. Prins. “Sambamba: fast processing of NGS alignment formats”. In: *Bioinformatics* 31.12 (June 2015). 25697820[pmid], pp. 2032–2034. issn: 1367-4811. doi: 10.1093/bioinformatics/btv098. url: https://www.ncbi.nlm.nih.gov/pubmed/25697820.

[17] G. G. Faust and I. M. Hall. “SAMBLASTER: fast duplicate marking and structural variant read extraction”. In: *Bioinformatics* 30.17 (Sept. 2014). 24812344[pmid], pp. 2503–2505. issn: 1367-4811. doi: 10.1093/bioinformatics/btu314. url: https://www.ncbi.nlm.nih.gov/pubmed/24812344.

[18] R. Poplin, P.-C. Chang, D. Alexander, S. Schwartz, T. Colthurst, A. Ku, D. Newburger, J. Dijamco, N. Nguyen, P. T. Afshar, S. S. Gross, L. Dorfman, C. Y. McLean, and M. A. DePristo. “A universal SNP and small-indel variant caller using deep neural networks”. In: *Nature Biotechnology* 36 (Sept. 2018). url: https://doi.org/10.1038/nbt.4235.

[19] D. C. Koboldt, Q. Zhang, D. E. Larson, D. Shen, M. D. McLellan, L. Lin, C. A. Miller, E. R. Mardis, L. Ding, and R. K. Wilson. “VarScan 2: Somatic mutation and copy number alteration discovery in cancer by exome sequencing”. In: *Genome Research* 22.3 (Feb. 2012), pp. 568–576. doi: 10.1101/gr.129684.111. url: https://doi.org/10.1101/gr.129684.111.

[20] Z. Lai, A. Markovets, M. Ahdesmaki, B. Chapman, O. Hofmann, R. McEwen, J. Johnson, B. Dougherty, J. C. Barrett, and J. R. Dry. “VarDict: a novel and versatile variant caller for next-generation sequencing in cancer research”. In: *Nucleic Acids Research* 44.11 (Apr. 2016), e108–e108. doi: 10.1093/nar/gkw227. url: https://doi.org/10.1093/nar/gkw227.

[21] K. Cibulskis, M. S. Lawrence, S. L. Carter, A. Sivachenko, D. Jaffe, C. Sougnez, S. Gabriel, M. Meyerson, E. S. Lander, and G. Getz. “Sensitive detection of somatic point mutations in impure and heterogeneous cancer samples”. In: *Nature Biotechnology* 31 (Feb. 2013), 213 EP -. url: https://doi.org/10.1038/nbt.2514.

[22] S. M. E. Sahraeian, R. Liu, B. Lau, K. Podesta, M. Mohiyuddin, and H. Y. K. Lam. “Deep convolutional neural networks for accurate somatic mutation detection”. In: *Nature Communications* 10.1 (Mar. 2019), p. 1041. issn: 2041-1723. doi: 10.1038/s41467-019-09027-x. url: https://doi.org/10.1038/s41467-019-09027-x.

[23] S. M. E. Sahraeian, L. T. Fang, M. Mohiyuddin, H. Hong, and W. Xiao. “Robust Cancer Mutation Detection with Deep Learning Models Derived from Tumor-Normal Sequencing Data”. In: *bioRxiv* (2019). doi: 10.1101/667261. eprint: https://www.biorxiv.org/content/early/2019/06/11/667261.1.full.pdf. url: https://www.biorxiv.org/content/early/2019/06/11/667261.1.

[24] D. P. Cooke, D. C. Wedge, and G. Lunter. “A unified haplotype-based method for accurate and comprehensive variant calling”. In: *Nature Biotechnology* (Mar. 2021). issn: 1546-1696. doi: 10.1038/s41587-021-00861-3. url: https://doi.org/10.1038/s41587-021-00861-3.

[25] E. Garrison and G. Marth. *Haplotype-based variant detection from short-read sequencing*. 2012. eprint: arXiv:1207.3907, [Accessed:11April2019].

[26] S. Kim, K. Scheffler, A. L. Halpern, M. A. Bekritsky, E. Noh, M. Källberg, X. Chen, Y. Kim, D. Beyter, P. Krusche, and C. T. Saunders. “Strelka2: fast and accurate calling of germline and somatic variants”. In: *Nature Methods* 15.8 (2018), pp. 591–594. issn: 1548-7105. doi: 10.1038/s41592-018-0051-x. url: https://doi.org/10.1038/s41592-018-0051-x.

[27] Z. Wei, W. Wang, P. Hu, G. J. Lyon, and H. Hakonarson. “SNVer: a statistical tool for variant calling in analysis of pooled or individual next-generation sequencing data”. In: *Nucleic Acids Research* 39.19 (Aug. 2011), e132–e132. doi: 10.1093/nar/gkr599. url: https://doi.org/10.1093/nar/gkr599.

[28] A. Wilm, P. P. K. Aw, D. Bertrand, G. H. T. Yeo, S. H. Ong, C. H. Wong, C. C. Khor, R. Petric, M. L. Hibberd, and N. Nagarajan. “LoFreq: a sequence-quality aware, ultra-sensitive variant caller for uncovering cell-population heterogeneity from high-throughput sequencing datasets”. In: *Nucleic*

Acids Research 40.22 (Oct. 2012), pp. 11189–11201. doi: 10.1093/nar/gks918. url: https://doi.org/10.1093/nar/gks918.

[29] FDA. *PrecisionFDA Truth Challenge*. 2019. url: https://precision.fda.gov/challenges/truth.

[30] FDA. *PrecisionFDA Truth Challenge V2: Calling Variants from Short and Long Reads in Difficult-to-Map Regions*. 2019. url: https://precision.fda.gov/challenges/10.

[31] L. X, K. Qiu, P. Liang, and P. D. "Speeding up large-scale next generation sequencing data analysis with pBWA". In: *Journal of biocomputing* 1 (Jan. 2012). doi: 10.4172/jbcg.1000101.

[32] A. Darling, L. Carey, and W. Feng. "The Design, Implementation, and Evaluation of mpiBLAST". In: *Proc Cluster World* 2003 (Dec. 2003).

[33] Y. Liu, B. Popp, and B. Schmidt. "CUSHAW3: Sensitive and Accurate Base-Space and Color-Space Short-Read Alignment with Hybrid Seeding". In: *PLOS ONE* 9.1 (Jan. 2014), pp. 1–9. doi: 10.1371/journal.pone.0086869. url: https://doi.org/10.1371/journal.pone.0086869.

[34] Apache. *Apache Arrow: A cross-language development platform for in-memory data [Accessed 29th Dec. 2019]*. 2019. url: https://arrow.apache.org/.

[35] Apache. *PySpark Usage Guide for Pandas with Apache Arrow [Accessed: 2nd April 2019]*. 2019. url: https://spark.apache.org/docs/latest/sql-pyspark-pandas-with-arrow.html.

[36] L. Jin. *Introducing Pandas UDF for PySpark*. 2018. url: https://bit.ly/3930obR.

[37] W. Shen, S. Le, Y. Li, and F. Hu. "SeqKit: A Cross-Platform and Ultrafast Toolkit for FASTA/Q File Manipulation". In: *PLOS ONE* 11.10 (Oct. 2016), pp. 1–10. doi: 10.1371/journal.pone.0163962. url: https://doi.org/10.1371/journal.pone.0163962.

[38] Apache. *Plasma In-Memory Object Store [Accessed 29th Dec. 2019]*. 2019. url: https://arrow.apache.org/blog/2017/08/08/plasma-in-memory-object-store/.

[39] FDA. *precisionFDA: A community platform for NGS assay evaluation and regulatory science exploration*. 2019. url: https://precision.fda.gov/.

[40] UCSC. *faSplit*. 2018. url: http://hgdownload.cse.ucsc.edu/admin/exe/linux.x86_64/.

[41] Illumina. *Illumina Cambridge Ltd. [Accessed 24th May 2019]*. 2012. url: ftp://ftp.1000genomes.ebi.ac.uk/vol1/ftp/phase3/data/NA12878/sequence_read/.

[42] T. E. N. A. (ENA). *Illumina 30X*. 2020. url: https://www.ebi.ac.uk/ena/browser/view/ERR194147.

[43] GIAB. *NHGRI Illumina 300X BAM*. 2020. url: ftp://ftp-trace.ncbi.nlm.nih.gov/giab/ftp/data/NA12878/NIST_NA12878_HG001_HiSeq_300x/NHGRI_Illumina300X_novoalign_bams/.

[44] UCSC. *UCSC hg19 (GRCh37)*. 2020. url: https://hgdownload.soe.ucsc.edu/goldenPath/hg19/bigZips/.

[45] SurfSara. *Cartesius: the Dutch supercomputer*. 2020. url: https://userinfo.surfsara.nl/systems/cartesius.

[46] Lustre. *Lustre parallel filesystem*. 2020. url: https://www.lustre.org/.

[47] Slurm. *Slurm workload manager*. 2020. url: https://www.schedmd.com/.

[48] A. CARROLL and N. THANGARAJ. *Evaluating DeepVariant: A New Deep Learning Variant Caller from the Google Brain Team*. 2017. url: https://bit.ly/3n4XtDT.

[49] T. Ahmad. *Standalone pre-processing on clusters*. 2021. url: https://bit.ly/3yC3QFf.

[50] P. Krusche. *Haplotype VCF comparison tools*. 2021. url: https://github.com/Illumina/hap.py.

5

Minimizing cluster data communication overhead

Prelude: *In the previous two chapters, we showed the benefits of the Apache Arrow (in-memory columnar data format) based ArrowSAM format for optimizing compute system resource utilization in variant calling pipelines. Arrow Flight is a submodule in the Apache Arrow project which provides a data transfer protocol for high performance, secure, parallel and cross-platform language support for bulk Arrow data transfers across the networks. In Section 5.1, we briefly introduce this protocol and provide detailed benchmarks for the client-server architecture and some recent examples where Arrow Flight is being used as a microservice for data transfer in distributed computing environments. In Section 5.2, we describe an Arrow Flight based cluster scalable variant calling solution, which does not require any big data or parallel programming language interface for genomics data processing in a cluster environment. We compare the runtime and communication performance results with MPI and Apache Spark based variant calling implementations. The content of this chapter is based on our papers, entitled "Benchmarking Apache Arrow Flight - A fast data transfer, query and microservices protocol" which is accepted for publication in ACM SIGPLAN "Benchmarking in the Data Center" workshop at PPoPP 2022, and the paper entitled, "Communication-Efficient Cluster Scalable Genomics Data Processing Using Arrow Flight" which is under review.*

5.1. Benchmarking Arrow Flight Performance

Moving structured data between different big data frameworks and/or data warehouses/storage systems often cause significant overhead. Most of the time more than 80% of the total time spent in accessing data is elapsed in serialization/de-serialization step. Columnar data formats are gaining popularity in both analytics and transactional databases. Apache Arrow, a unified columnar in-memory data format promises to provide efficient data storage, access, manipulation and transport. In addition, with the introduction of the Arrow Flight communication capabilities, which is built on top of gRPC, Arrow enables high performance data transfer over TCP networks. Arrow Flight allows parallel Arrow RecordBatch transfer over networks in a platform and language-independent way, and offers high performance, parallelism and security based on open-source standards. In this paper, we bring together some recently implemented use cases of Arrow Flight with their benchmarking results. These use cases include bulk Arrow data transfer, querying subsystems and Flight as a microservice integration into different frameworks to show the throughput and scalability results of this protocol. We show that Flight is able to achieve up to 6000 MB/s and 4800 MB/s throughput for DoGet() and DoPut() operations respectively. On Mellanox ConnectX-3 or Connect-IB interconnect nodes Flight can utilize upto 95% of the total available bandwidth. Flight is scalable and can use upto half of the available system cores efficiently for a bidirectional communication. For query systems like Dremio, Flight is order of magnitude faster than ODBC and turbodbc protocols. Arrow Flight based implementation on Dremio performs 20x and 30x better as compared to turbodbc and ODBC connections respectively. We briefly outline some recent Flight based use cases both in big data frameworks like Apache Spark and Dask and remote Arrow data processing tools. We also discuss some limitations and future outlook of Apache Arrow and Arrow Flight as a whole. Test codes and scripts are available at
https://github.com/abs-tudelft/time-to-fly-high

5.1.1. Introduction

Transferring data between databases/data storage systems and client programs in bulk amounts for machine learning applications or statistical analysis, is a common task in data-science. This operation is rather expensive as compared to subsequent operations and becomes even more expensive when the data storage server runs on a different machine or in a cloud environment. Open-source data science developers/researchers and organizations heavily rely on the Python/R-based data-science eco-system. Apache Parquet, ORC, Avro, and HDFS are commonly used binary formats to store data in compressed form other than text based CSV format. Data serialization and de-serializations on different data processing pipelines (e.g., converting to Pandas Dataframes) built in this eco-system add an additional overhead before actual data can be processed. If data has to be transferred from a remote DBMS server using the DBMS network protocol to these applications, it becomes more expensive due to: i. reading from row-oriented

DBMS, ii. transferring via slower ODBC/JDBC network protocols, iii. converting it to required columnar format. So converting row-store data to columnar format is always a major source of inefficiency in data analytics pipelines [1]. As these formats are designed to store imputable data structures (write-once and read-many), because of this reason they are supposed to be helpful in data analytics workloads only and are not susceptible for transactional workloads. Conventionally, row-store DBMS has been used for OLTP workloads, however recent work by SAP HANA [2] paves the way to bring up column-oriented databases to the mainstream by introducing a highly scalable and efficient query processing engine for both transactional and analytics workloads. TiDB [3] is an open-source example of such a hybrid database system, that supports both transactional and analytical workloads. It is both distributed and MySQL compatible, featuring horizontal scalability, strong consistency, and high availability.

Apache Arrow provides an open-standard unified in-memory and columnar data format. It alleviates the need of serialization/de-serialization of data through a common format and by providing interfaces for different languages, which makes zero-copy inter-process communication possible. Although Arrow targets mainly OLAP (read-only) workloads, OLTP workloads can still benefit from it. Arrow Flight a submodule in Arrow project provides a protocol to implement a service which can send and receive Arrow (RecordBatches) data streams over the network.

In this work, we discuss the current state-of-the-art for Arrow Flight in terms of development and its applications. We benchmark the performance of Arrow Flight on both the client-server model as well as on the cluster environment and examine the actual speed and bottlenecks in Arrow data transfer, query execution and in microservices usage in different distributed big-data/machine learning frameworks.

The reminder of this paper is organized as follows: In section "Background", we discuss the Apache Arrow and Arrow Flight internals and architecture in details, followed by a "Data Transfer Benchmarks" sections, where Arrow Flight localhost and client-server benchmarks are discussed. We then describe query subsystem and Arrow Flight as microservice integration into some data analytic frameworks in "Use Cases" section. In section "Future Outlook" a short future perspective is given. At the end in "Conclusion" section we conclude this work by outlining some future approaches and use cases.

5.1.2. Background

In this section, we outline the architectural and design aspects of Apache Arrow and its APIs, particularly Arrow Flight, in detail.

Apache Arrow: Apache Arrow [4] intends to become a standard columnar format for in-memory data analytics. Introduced in 2015, Arrow provides cross-language interoperability and IPC by supporting different languages, C, C++, C#, Go, Java, JavaScript, MATLAB, Python, R, Ruby, and Rust. Arrow also provides support for heterogeneous platforms in the form of rapids.ai for GP-GPUs and Fletcher for FPGA systems [5]. Apache Arrow is increasingly extending its ecosystem by supporting different APIs (e.g., Parquet, Plasma Object Store, Arrow

Table 5.1 Example table stored as an Arrow RecordBatch

X	Y	Z
555	"Arrow"	5.7866
56565	"Data"	0.0
null	"!"	3.14

Table 5.2 Arrow Buffers layout for data in Table 5.1

	Arrow Buffers for:				
	Field X		Field Y		Field Z
Index	Validity (bit)	Values (Int32)	Offsets (Int32)	Values (Utf8)	Values (Double)
0	1	555	0	A	5.7866
1	1	56565	5	r	0.0
2	0	null	9	r	3.14
3				o	
4				w	
5				D	
6				a	
7				t	
8				a	
9				!	

Table 5.3 Schema for RecordBatch in Table 5.1

Field X: Int32 (nullable), Field Y: Utf8, Field Z: Double

Compute, etc.) and many open-source libraries/tools are integrating Arrow inside them for efficient data manipulation and transfer. For example, TensorFlow has recently introduced the TensorFlow I/O [6] module to support the Arrow data format, the Dremio big data framework is built around the Apache Arrow ecosystem, pg2arrow (a utility to query PostgreSQL relational database), turbodbc which supports queries in Arrow format, etc.

Arrow stores data in contiguous memory locations to make the most efficient use of CPU's cache and vector (SIMD) operations. In the Arrow format, data entries (records) are stored in a table called a RecordBatch. An example of a RecordBatch with three records (rows) and three fields (columns) is shown in Table 5.1. As shown in Table 5.2, each field in the RecordBatch table is stored in a separate memory region in a manner that is as contiguous as possible in memory. This memory region is called an Arrow Field or Array which can store data of different types—i.e., int, float, UTF8 characters, binary, timestamps, lists and nested types. Depending on the data types, fields can have multiple Arrow Buffers to store extra information about the data, such as a validity bit for nullable data types, or offsets in the case of variable-sized lists. Through this approach, accessing data from random locations and in parallel with a minimum amount of pointers traversing becomes possible. This approach makes Arrow less efficient particularly in large write-updates of variable length strings which is a point of concern for using Arrow in transactional workloads.

Each RecordBatch contains metadata, called a schema, that represents the data types and names of stored fields in the RecordBatch. Table 5.3 shows the schema of the example Arrow RecordBatch shown in Table 5.1.

Arrow Flight: Arrow Flight [7] provides a high performance, secure, parallel and cross-platform language support (using the Apache Arrow data format) for bulk data transfers particularly for analytics workloads across geographically distributed networks. Using Apache Arrow as standard data format across all languages/frameworks as well as on the wire, Arrow Flight data (Arrow RecordBatches) does not require any serialization/de-serialization when it crosses process boundaries. As Arrow Flight operates directly on Arrow RecordBatches without accessing data of individual rows as compared to traditional ODBC/JDBC interfaces, it is able to provide high performance bulk operations. Arrow Flight supports encryption out of the box using gRPC's built in TLS/OpenSSL capabilities. Simple user/password authentication scheme is provided out-of-the-box in Arrow Flight and provides extensible authentication handlers for some advanced authentication schemes like Kerberos.

In basic Arrow Flight communication, a client initiates the communication by sending the `GetFlightInfo()` command to the server. In case of a successful connection, the server replies with available Flights by sending back `FlightInfo` information, which contains so-called `Ticket`s that define locations (or `Endpoint`s) of streams of RecordBatches at the server side. Then, the `DoPut()` command is used by the client to send a stream of RecordBatches to the server, and the `DoGet()` command is used by the server to send a stream back to the client. Both these commands are initiated by the client. Figure 5.1(a)

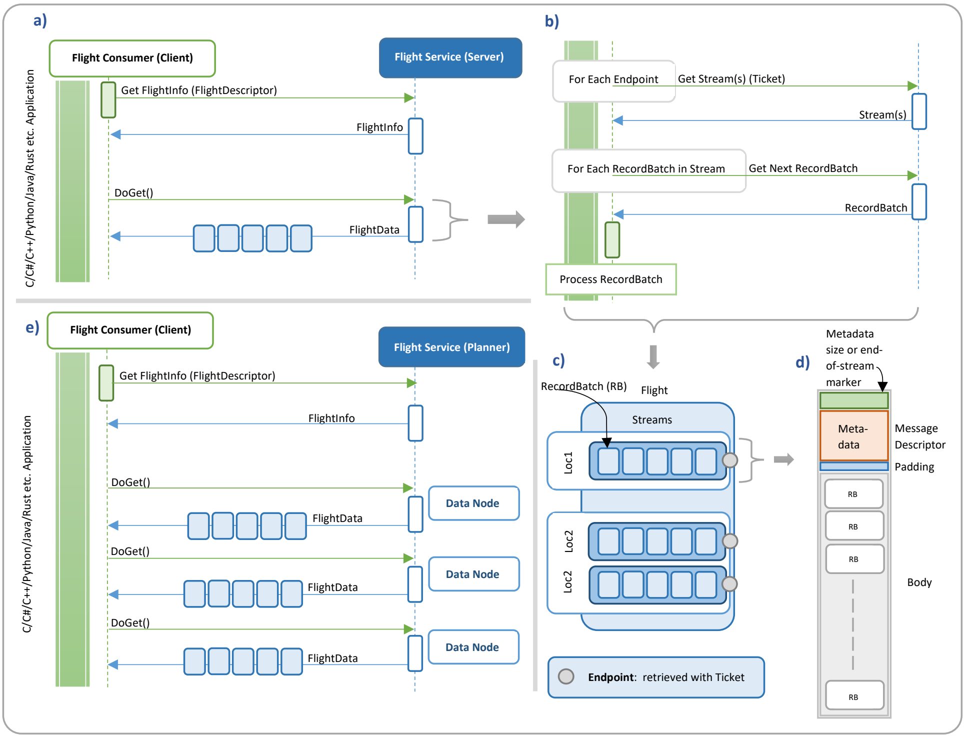


Figure 5.1 a) Arrow Flight client-server communication protocol dataflow diagram for DoGet() operation. b) In depth Flight streams communication dataflow for accessing RecordBatches in a stream. c) Flight streams endpoint. d) Inter-process communication format. e) Arrow Flight cluster communication protocol dataflow diagram with multiple nodes and a single planner node for DoGet() operation.

shows the data flow protocol diagram for an example Flight communication with the `GetFlightInfo()` and `DoGet()` commands. In Figure 5.1(b), the client uses the `GetStream` command to request one or more streams of RecordBatches by calling their `Ticket` information. Figure 5.1(c) shows the internal structure of Flight communication. Figure 5.1(d) shows the Flight protocol description within each stream, which contains the stream metadata and RecordBatches.

Flight services can handle multiple Flight connections in a cluster environment and can differentiate between them using a Flight descriptor, which can define the composition of Flight connections with batch size, and either file name or SQL query command as shown in Figure 5.1(e).

Distributed Columnar-store and Analytics

Relational databases are optimized for transactional workloads, which makes them less efficient to support the needs of modern analytics applications. Querying billions of rows on demand from a row-oriented database for analytics and statistical purposes becomes a bottleneck in real-time column-oriented data analytics workloads. Moreover, production-ready analytics workloads and ML pipelines mostly use public clusters and cloud computing infrastructures for a couple of reasons including security, scalability, high-availability, lower costs and on-demand easy to deploy functionality. All major cloud service providers present their own distributed datastores like Google (Cloud SQL for OLTP and BigQuery for OLAP systems), Amazon (AWS Redshift and AWS S3) and Azure (Cosmos DB and Synapse Analytics) for both analytics and SQL. Apache Hive, Dremio, Presto and Apache Impala are a couple of BI/data science SQL based engines built to communicate with distributed datasets using different storage formats. The support of universal storage formats (like HDFS, ORC, CSV, Parquet) makes these systems flexible to export data in any form and to any system for further processing. For such a distributed data-store environment, it essential to provide high-throughput methods to communicate large datasets between systems. The Arrow format also supports local, Parquet, HDFS and S3 distributed file systems, which makes Arrow Flight an important differentiator for Arrow-based applications.

DB-X Data Export to External tool: As reported in [1], the authors evaluate the data export (to an external tool) performance of DB-X. They compared four different export methods, (1) client-side RDMA, (2) Arrow Flight RPC, (3) vectorized wire protocol from [8], and (4) row-based PostgreSQL wire protocol. They used the TPC-C `ORDER_LINE` table with 6000 blocks (approximately 7 GB total size) on the server. By varying the % of frozen blocks in DB-X they study the impact of concurrent transactions on export speeds. Figure 5.2 shows when all the blocks are frozen, RDMA saturates the bandwidth while Flight uses up to 80% of the available bandwidth. When the system has to materialize every block, the performance of Arrow Flight drops to be equivalent to the vectorized wire protocol. RDMA performs slightly worse than Arrow Flight with a large number of hot blocks, because Flight has the materialized block in its CPU cache, whereas the NIC bypasses this cache when sending data. Overall the experiment shows that (de)-serialization is the main bottleneck in achieving better data export speeds in DBMS. Using a common data format in DBMS like Arrow can boost the export speeds in-conjunction with the Arrow Flight wire protocol.

5.1.3. Data Transfer Benchmarks

Bulk data transfers over long-haul networks has become an integral part of modern data science applications. Reading and extracting required data from remote datasets through remote data services like ODBC or JDBC is inefficient and lacks support for current applications and frameworks. Although in the past decade, file-based (text formats like CSV, JSON and binary formats like Avro, ORC and Parquet) data warehousing has become popular, still raw data needs serialization/de-serialization to a particular format when accessed/used by different applications on remote/local servers. With Arrow Flight, a unified Arrow columnar data format can be used, which provides both over-the-wire data representation as well as a public API for different languages and frameworks. This in

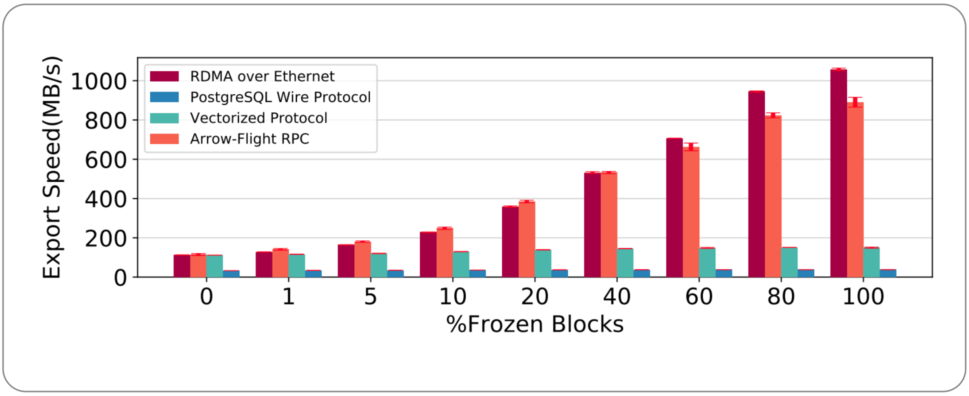


Figure 5.2 Data Export – Measurements of export speed with different export mechanisms in DB-X, varying % of hot blocks [1].

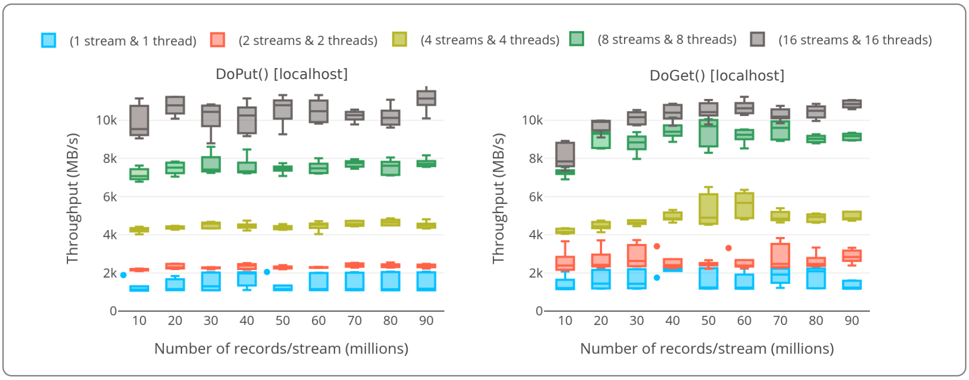


Figure 5.3 Arrow Flight DoPut() and DoGet() throughput with multiple stream/threads with varying number of records per stream (10-90 million) on a localhost.

turn eliminates much of the serializations overheads associated with data transport.

Evaluation system: Arrow Flight based bulk data transfer benchmarks in this section are executed on the SurfSara Cartesius [9] HPC cluster (part of the Dutch national supercomputing infrastructure). Each CPU-only node is equipped with a dual socket Intel Xeon Processor (E5-4650) running at 2.7 GHz. Each processor has 32 physical cores with support of 64 hyper-threading jobs. A total of 256-GBytes of DDR4 DRAM with a maximum of 59.7 GB/s bandwidth is available for each node. All nodes are connected through Mellanox ConnectX-3 or Connect-IB InfiniBand adapter providing 4×FDR (Fourteen DataRate) resulting in 56 Gbit/s inter-node bandwidth.

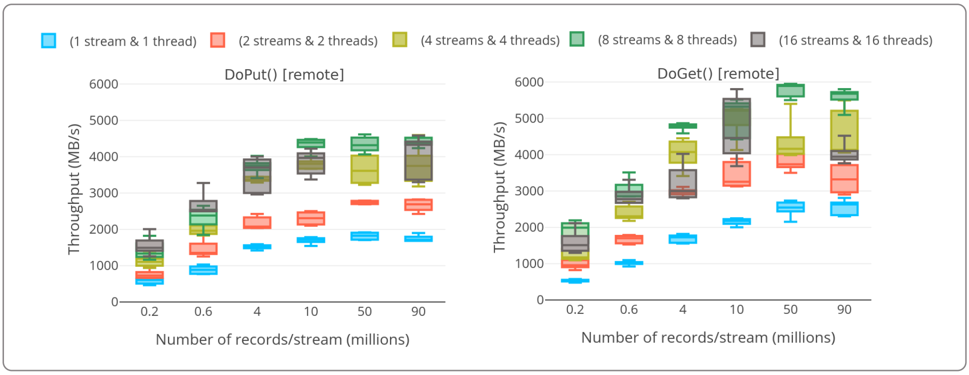


Figure 5.4 Arrow Flight DoPut() and DoGet() throughput with multiple stream/threads with varying number of records per stream (0.2-90 million) on a remote client-server nodes connected through a Mellanox ConnectX-3 or Connect-IB InfiniBand adapter.

Client-Server Microbenchmarks

To measure the absolute speed and performance of Arrow Flight, we use the Flight built-in performance benchmark written in C++ in both localhost and on a network in a client-server setting. In localhost, a loopback network interface

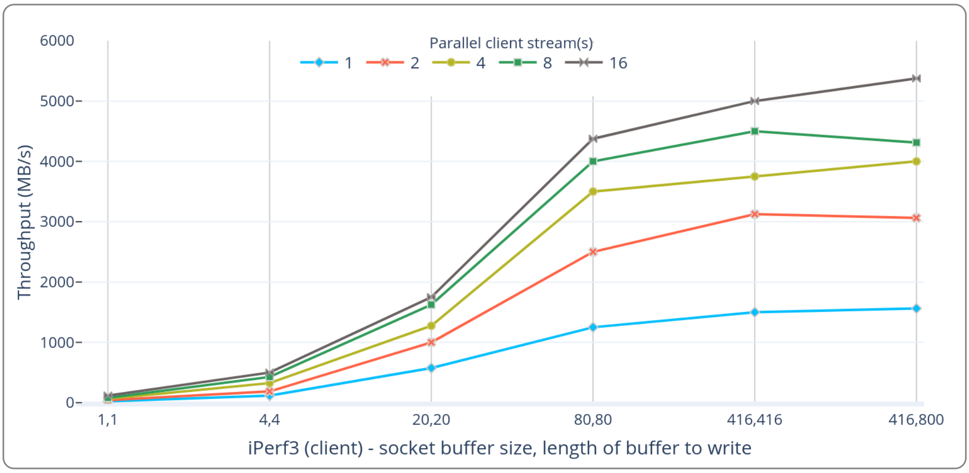


Figure 5.5 iPerf3 based client/server benchmarking of TCP data send/receive overall transfer throughput on a Mellanox ConnectX-3 or Connect-IB InfiniBand adapter based system.

is established on a single computer node. Usually, in client-server model server controls the communication between associated client(s) over the network. Figure 5.3 shows throughput variation of Arrow Flight data transport for localhost while Figure 5.4 shows throughput for client-server settings. We use 1, 2, 4, 8

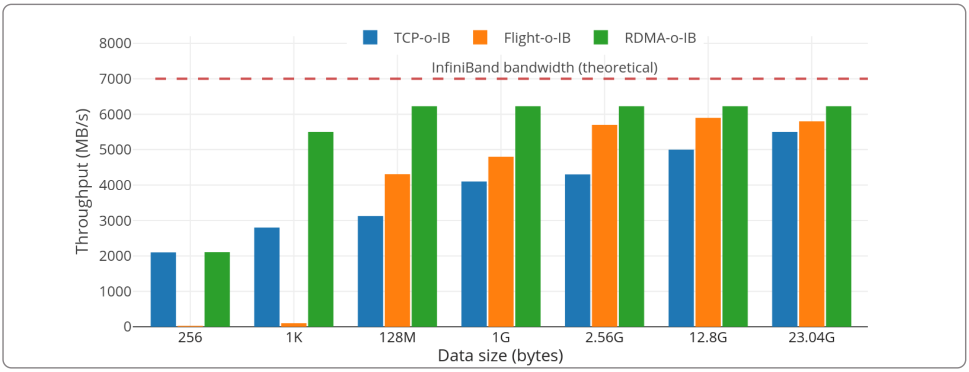


Figure 5.6 Throughput comparison of IPoIB using iPerf3, Flight-over-IB and RDMA (Infinaband) on a Connect-IB InfiniBand adapter based client-server remote system.

and 16 streams in parallel with each stream having 10-90 million records. Each record contains 32 bytes. On localhost, both `DoPut()` and `DoGet()` functions give a throughput in the order of 1GB/s for single stream up to 10GB/s with 16 streams in parallel. As the localhost processor has 16 physical cores on two sockets with support of 32 hyper-threading jobs. So Arrow Flight performance shows a significant increase in throughput when more parallel streams are employed. We also observe that increasing the parallel streams more than 16 decreases the overall performance. We run the client-server benchmark in a network [9] in which every node has a Mellanox ConnectX-3 or Connect-IB (Haswell thin nodes) InfiniBand adapter providing 4 × FDR (Fourteen Data Rate) resulting in 56 Gbit/s inter-node bandwidth. We see the same trend in this remote data transfer setting with throughput increasing from 1.2GB/s to 1.65GB/s for `DoPut()` while `DoGet()` achieves 1.5GB/s to 2GB/s throughput with up to 16 streams in parallel.

To compare the throughput of Arrow Flight with other common communication protocols, Figure 5.6 measures the throughput of Flight over InfiniBand (Flight-o-IB) and two other communication protocols on the same network for remote client-server communication: 1. the TCP protocol over InfiniBand (TCP-o-IB), commonly used for long-haul data communication, and 2. RDMA over InfiniBand (RDMA-o-IB) protocol, commonly used for high-throughput cluster-based communication. To measure TCP throughput, we use the iPerf3 [10] network performance measurement tool with multiple parallel streams, which is able to measures raw TCP throughput with minimal overhead. For RDMA throughput, we use the `ib_write_bw` (InfiniBand write bandwidth) tool which is part of the Perftest Package [11]. Figure 5.6 shows that RDMA is able to achieve a high throughput of 6.2GB/s (close to the theoretical max of 7GB/s) for a wide range of data sizes. TCP, on the other hand, has a low throughput of about 2GB/s for small data sizes (256B) that increases slowly as the data size increases, consistently suffering from high overhead for a wide range of data sizes. In contrast, Flight has extremely low bandwidth for very small data sizes of up to 1KB, but then

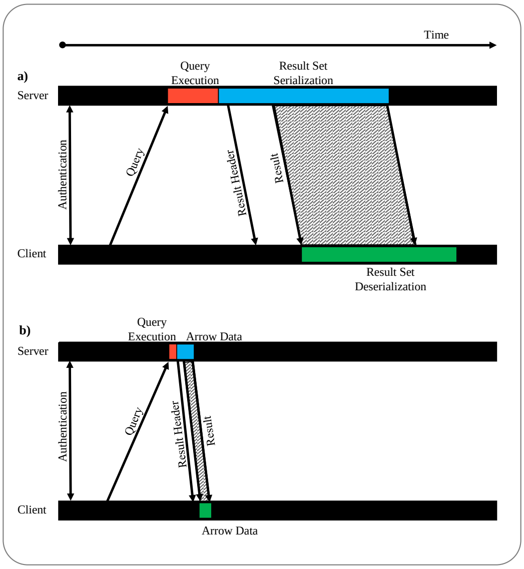


Figure 5.7 A typical client-server communication for query execution [8]. (a) Without Flight: a large amount of time spent in (de)-serialization of the result set is shown. (b) With Flight: the total time spent in query execution on Arrow data with Arrow Flight based communication eliminates any (de)-serialization overhead.

consistently outperforms TCP for larger sizes and is able to achieve about 95% of the RDMA bandwidth (or more than 80% of the maximum achievable bandwidth) for data sizes of 2.6GB or larger. This shows the capabilities of Flights to ensure high throughput for bulk data transfers that is comparable to high-throughput protocols such as RDMA over InfiniBand, while retaining the benefits of ease of programmability, security, and allowing access to a wide range of web-based services.

In addition, the figures show that Flight allows improving the throughput by increasing the number of parallel streams. However, this is not the case for TCP, as increasing the number of streams results in more network congestion and a slight reduction in throughout.

5.1.4. Use Cases

This section presents some common use cases of Arrow Flight related to query data transfer from a remote data service, and to Arrow Flight usage in big data frameworks for distributed data transfers in a cluster to be consumed in microservices.

Query Subsystem

Transferring big amounts of data from a local/remote server to a querying client is a common task in both analytics (statistical analysis or machine learning applications) and transactional database systems. As described in [8], a typical client-server data communication scenario is shown in Figure 5.7, where the communication time is heavily influenced by data serialization overhead.

Arrow Flight provides a standard in-memory unified columnar data format. Exporting Arrow tables to any client language interface avoids (de)-serialization overhead. Better columnar compression techniques and parallel Arrow streams transfer make Arrow Flight ideal for efficient big data transfers.

In this section, we focus on Dremio, an production grade Arrow native analytics framework. We measure the performance metrics of different client side protocols (ODBC and turbodbc) for querying on the Dremio remote client-server and compare the results with Arrow Flight as shown in Figure 5.8. We also look at two systems under development: Arrow Datafusion (an Arrow Flight based client-server query API), and FlightSQL (a native SQL API for Arrow data).

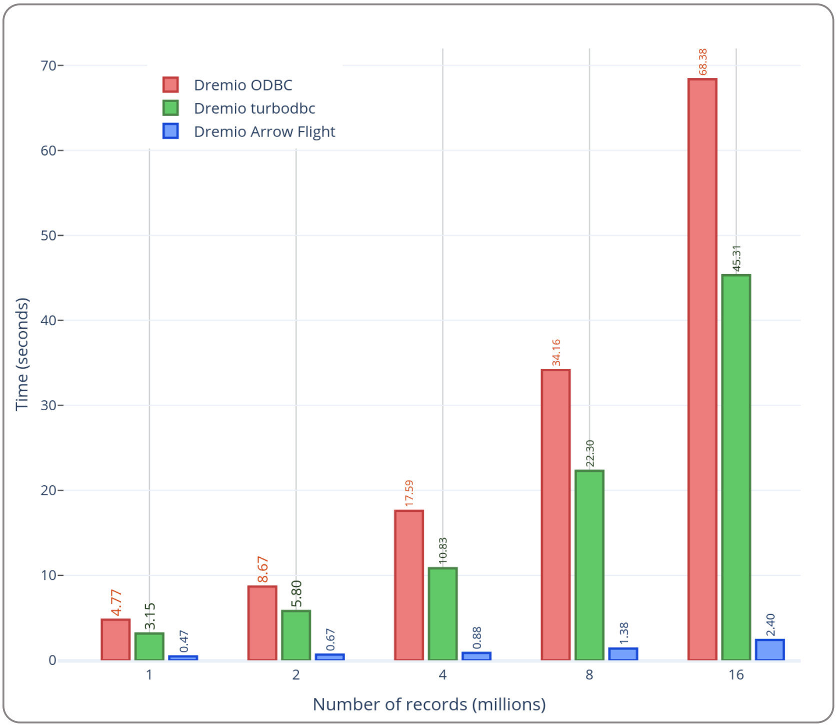


Figure 5.8 Total time spent in querying NYC Taxi dataset on a remote Dremio client-server nodes with varying number of records (1-16 millions) through ODBC, turbodbc and Flight connections.

Dremio - ODBC: Dremio provides a custom ODBC driver for different client types. We used a Linux based Dremio ODBC driver and used it with the pyodbc Python API to query the NYC Taxi database (in parquet format) from a Dremio server running remotely in a cluster.

Dremio - turbodbc: We also used the Dremio ODBC driver to connect with a Dremio client through the turbodbc Python API. Here too, we queried the NYC Taxi database (in parquet format) from a Dremio server running remotely in the same cluster.

Dremio - Flight: Dremio offers client and server Flight endpoint support for Arrow Flight connections that is also authentication OAuth2.0 compliant. Moreover this implementation provides TLS encryption to establish an encrypted connection.

The runtime performance comparison results for all three methods on a single select query are shown in Figure 5.8. Arrow Flight based implementation on Dremio performs 20x and 30x better as compared to turbodbc and ODBC connections respectively.

Data-Fusion - Flight: DataFusion is an in-memory, Arrow-native query engine implemented in Rust. Though this framework is in its initial phases of development, it does support SQL queries against iterators of RecordBatch in both CSV and Parquet file formats. Both the Arrow Flight client and server implementations are available. Figure 5.9 shows the results we obtained by running the Arrow Flight client-server benchmark provided in Data-Fusion repository. We converted NYC Taxi dataset used in previous Dremio demo to Parquet format and query the same dataset for specific elements in each iteration.

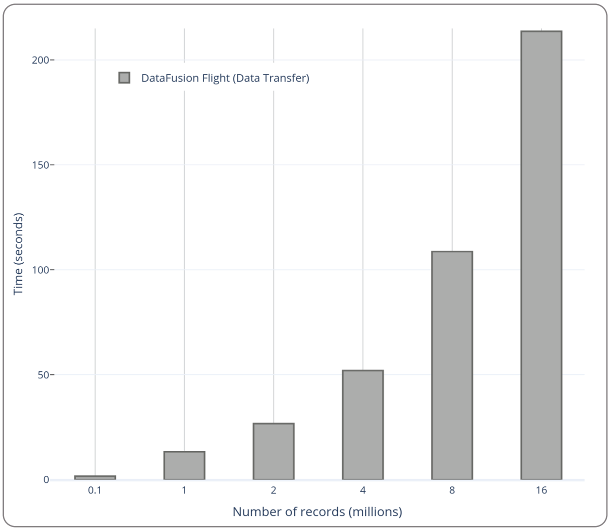


Figure 5.9 Total time spent in querying NYC Taxi dataset on a remote Arrow Flight based client-server nodes implemented in Data-fusion project with varying number of records (0.1-16 millions).

Apache Arrow - FlightSQL: FlightSQL is a new proposal being implemented by the Apache Arrow community to become a standard way of accessing Arrow data via SQL-like semantics over Flight. The main idea of this framework is to use ODBC and JDBC data access best practices while maintaining the high throughput facilitated by Arrow Flight.

Microservices Integration

Arrow Flight can be integrated into data transfer and remote data analytics microservices for efficient and parallel processing of Arrow columnar data using many different frameworks like Dask and Spark, as discussed next.

Flight Data Microservice - Apache Spark [12]: This Arrow Flight based microservice implementation is an early prototype test to showcase the reading of columnar Arrow data, reading in parallel many Flight endpoints as Spark partitions, this design uses the Spark Datasource V2 API to connect to Flight endpoints. Figure 5.10 shows performance comparisons in terms of total time for default JDBC, serial flight, parallel flight and parallel flight with 8 nodes. This test returns n rows to Spark executors and then performs a non-trivial calculation on them. This test was performed on a 4x node EMR with querying a 4x node Dremio AWS Edition (m5d.8xlarge) by the developer.

Flight Data Microservice - Apache Spark/TensorFlow Clients [13]: A basic Apache Arrow Flight data service with Apache Spark and TensorFlow clients has been demonstrated. In this demo a simple data producer with an InMemoryStore allows clients to put/get Arrow streams to an in-memory store. Existing PySpark DataFrame partitions are mapped by a Spark client to produce an Arrow stream of each partition which are put under the FlightDescriptor. A PyArrow client reads these streams and convert them into Pandas Dataframes. Similarly, a TensorFlow client reads each Arrow stream, one at a time, into an ArrowStreamDataset so records can be iterated over as Tensors [13].

XGBatch - Pandas/Dask [14]: ML model deployment generally consists of two phases. First, models are trained and validated with existing datasets to uncover pattern and correlations within the data. Then, the best performing trained model is applied to new datasets, to perform various tasks, such as predicting the probability scores in the case of classification problems, or estimating averages in the case of regression problems [15]. In production environments, ML based applications usually have separate deployment methods for real time model needs (e.g., an API or gRPC service, etc.) vs batch scoring (e.g., some form of Spark or Dask based solution) [14]. Real time use cases need low latency for processing millions of records each row at a time, while batch processes need to take advantage of modern hardware features like multi-cores, vectorization, accelerators (GPUs/FPGAs) and high throughput interconnects on cluster environments to process and transfer the large amount of data quickly. XGBATCH as shown in Figure 5.11 uses Apache Arrow's Flight framework (which is built on the gRPC protocol under the hood) to stream batches of data to the scoring service, which in-turn scores it as a batch, and finally streams the batch back to the client. Using Flight ensures low latency for real time use cases, as well as an efficient method for scoring large batches of data.

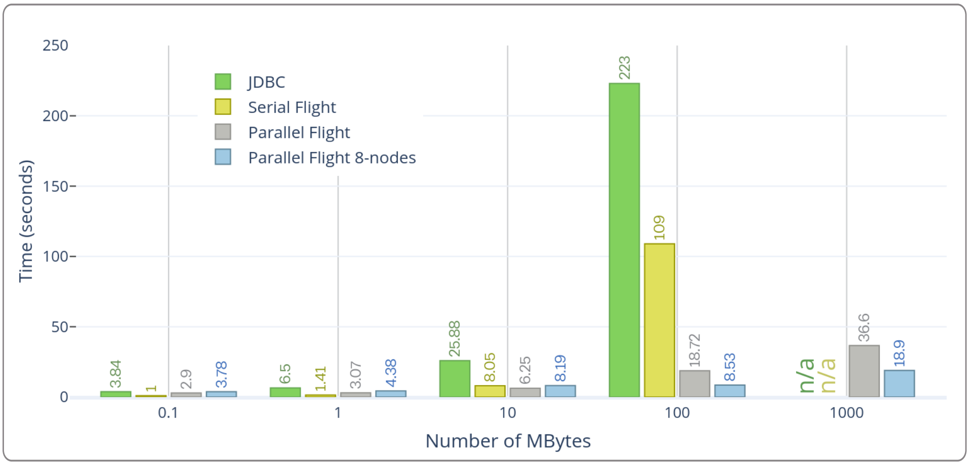


Figure 5.10 Performance results of Apache Arrow Flight endpoints integration in Apache Spark using the Spark Datasource V2 interface. The results show the total time spent in Spark default JDBC, serial Flight, parallel Flight and parallel Flight with 8-node connections.

FlightGrid/PyGrid - AI Models Training [16]: PyGrid is a peer-to-peer platform for secure, privacy-preserving and decentralized data science and analytics. Data owners and scientists can collectively train AI models using the PySyft framework. In PyGrid data-centric federated learning (FL) use cases, a lot of data movement between domain and workers network is involved. In a FlightGrid implementation for a simple network using mnist dataset with batch size 1024 and pre-trained model with Arrow data format on Arrow Flight nodes shows more than 5x speedup with the same accuracy as compared to regular grid data.

The Mesh for Data platform - Arrow/Flight module [17]: The Mesh for Data is a cloud-native platform to control the data usage within an organization premises. It provides a platform to unify data access, governance and orchestration, enabling business agility while securing enterprise data. The arrow-flight-module (AFM) for The Mesh for Data brings enforcement of data governance policies to the world of Apache Arrow Flight for fast and efficient data movement and analytics within applications to consume tabular data from the data sources. Currently, AFM provides support for a couple of different file-systems, formats and queries for Arrow datasets.

5.1.5. Future Outlook

Currently Flight service operations rely only on TCP data transport layer. Using gRPC to coordinate get and put transfers on protocols other than TCP like RDMA have huge potential to speed up bulk data transfers over the networks with RDMA support. As shown in [1] for some typical databases, the data export speed to some external applications while the majority of the blocks is in a frozen state can utilize up to 80% of total available network bandwidth. This result suggests that

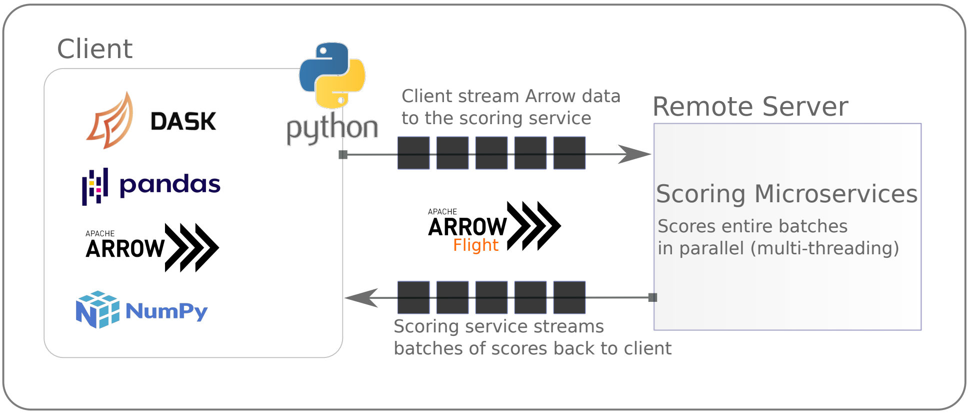

Client
DASK
pandas
APACHE ARROW
NumPy
python
Client stream Arrow data to the scoring service
APACHE ARROW Flight
Scoring service streams batches of scores back to client
Remote Server
Scoring Microservices
Scores entire batches in parallel (multi-threading)

Figure 5.11 A remote scoring microservice using Arrow data batches and communicating via Arrow Flight.

bypassing the network stack for Arrow bulk data transfers via RDMA can easily saturate high bandwidths networks. The Flight SQL proposal [18] which is being implemented paves the way for client-server databases to directly communicate with SQL-like semantics. This feature will enable browsing database metadata and execution of queries while transferring data streams with Arrow Flight.

In the context of distributed systems, many different distributed columnar databases and query engines also propose to integrate an Arrow Flight layer support for data export to external applications/transfer bulk data in Arrow supported frameworks. In addition, many distributed AI/ML training and inference workloads [6, 14] are also being equipped with Arrow Flight functionality.

Limitations of this work: This work is an early of what is possible with Arrow format and Arrow Flight as an Arrow data transfer, querying and microservice context. Arrow APIs including Arrow Flight are under heavily development process for both new features addition and performance improvements. At the same time all the projects discussed in this article are also under development. We believe in coming months these projects will be matured enough to be integrated into existing frameworks for both better performance and scalability.

5.1.6. Conclusion

Apache Arrow is a columnar in-memory data format which provides cross-language support for data analytic applications and frameworks. It enables fast data movement within big data frameworks eco-system by avoiding (de)-serialization overheads. Arrow Flight is gRPC based framework which provides high speed data communication services for Arrow data transfer over the networks. In this article, we demonstrated and benchmarked a couple of Arrow Flight use cases. For bulk Arrow data transfer we benchmarked the throughput on both local and remote hosts with varying batch sizes on a 7000 MB/s inter-node bandwidth. The maximum 1650 MB/s throughput achieved for DoPut() while DoGet() achieves upto 2000 MB/s throughput with upto 16 streams in parallel on remote hosts. On local machine Arrow Flight achieves upto 10K MB/s throughput. In genomics pipeline, the distributed regions specific chromosomes sorting of ArrowSAM data achieves upto 500 MB/s throughput. Note that in this particular scenario, all nodes are connected through Flight endpoints and sending/receiving Arrow RecordBatch streams at the same time in parallel. We also included the results of DB-X bulk data export speeds of different (client-side RDMA, Arrow Flight, vectorized wire protocol and row-based PostgreSQL wire protocol) protocols, where Flight protocol uses nearly 80% of total available bandwidth in case all blocks are frozen. By comparing the results of different data querying APIs like ODBC, turbodbc and Flight on a Dremio client also shows a significant performance/data transfer time improvement when accessing/querying somehow big size datasets. Arrow Flight based implementation on Dremio performs 20x and 30x better as compared to turbodbc and ODBC connections respectively. Rust based Datafusion Flight API also provides client-server implementation for SQL querying on CSV and Parquet data over Flight. Moreover, we also analysed some microservices uses cases like Apache Spark and TensorFlow clients to put/get Arrow data streams in parallel where Flight can be used as a fast Arrow data transfer layer to speedup the analytical processes on batches. Reading multiple Flight endpoints in parallel as Spark partitions in a multi-node cluster as compared to existing serial JDBC approach in Spark improves the performance by many folds. Batch scoring/processing and remote ML models training/testing on single as well as multi-node cluster environments on Arrow data through Flight has potential to improve the performance of existing algorithms by an orders-of-magnitude.

5.2. Arrow Flight based variant calling workflow

Current cluster scaled genomics data processing solutions rely on big data frameworks like Apache Spark, Hadoop and HDFS for data scheduling, processing and storage. These frameworks come with additional computation and memory overheads by default. It has been observed that scaling genomics dataset processing beyond 32 nodes is not efficient on such frameworks.

To overcome the inefficiencies of big data frameworks for processing genomics data on clusters, we introduce a low-overhead and highly scalable solution on a SLURM based HPC batch system. This solution uses Apache Arrow as in-memory columnar data format to store genomics data efficiently and Arrow Flight as a network protocol to move and schedule this data across the HPC nodes with low communication overhead.

As a use case, we use NGS short reads DNA sequencing data for pre-processing and variant calling applications. This solution outperforms existing Apache Spark based big data solutions in term of both computation time (2x) and lower communication overhead (more than 20-60% depending on cluster size). Our solution has similar performance to MPI-based HPC solutions, with the added advantage of easy programmability and transparent big data scalability. The whole solution is Python and shell script based, which makes it flexible to update and integrate alternative variant callers. Our solution is publicly available on GitHub at https://github.com/abs-tudelft/time-to-fly-high/genomics

5.2.1. Introduction

Due to massively parallel sequencing methods used in high throughput sequencing (HTS) technologies are making their way from research to the field in a wide range of applications ranging from clinical diagnostics to agriculture research. Next Generation Sequencing (NGS) technologies like Illumina short-reads (a couple of hundred bases), can produce high throughput and higher depth DNA sequence coverage at low cost. Similarly, longer read third generation sequencing technologies are also emerging as a more competitive alternative in terms of cost and throughput with improving accuracy as compared to NGS. They can produce reads of up to hundreds of kilobases (kbps). Depending on the experiment design, the need of sequencing coverage varies [19]. A typical 300x coverage human genome dataset size exceeds 2 TBytes [19]. Processing this amount of data on a single computing machine can take multiple days to complete.

High-throughput sequencing technologies are also enabling cancer diagnoses and treatment beyond histopathology and traditional standard-of-care therapies. Molecular and genomic profiling for patients and tumours at time of diagnosis help in improving diagnostic accuracy, better predict outcome, and personalize therapy [20]. Sequencing coverage influence both the accuracy and sensitivity of such genomics analysis. In pediatric brain-tumor studies [20], more than 200x coverage for the tumor sample, and more than 100x coverage for the normal sample are collected for better focus on the concerned tumors.

In the coming years, as sequencing becomes a normal practice for human health and other types of research, single node compute resources to any organization will not be adequate to fulfill the sequencing requirements. The increased need for data processing will lead to use cluster scaled solutions and outsourcing genomics computations to external private and public cloud services on data centers.

Genonmics data processing pipelines (e.g, short-variants, structural variants and copy-number variants discovery) involve many computational processing steps. Sequence alignment and variant calling are two important steps while intermediate steps like sorting, duplicates removal and base quality score recalibration which use row-based SAM/BAM format to store the outcome of these algorithms on I/Os. Generally genomics data formats (FASTQ/SAM/BAM) permit independent compute and analytic operations on a granular level, i.e., even smaller chunks can be processed without any dependency issues. This eventually helps to run genome analysis algorithms on multiple data chunks in parallel. Halvade [21], which uses the Hadoop MapReduce API, while ADAM [22] and SparkGA2 [23] use the Apache Spark framework and HDFS as a distributed file system are few examples of frameworks which use big data frameworks to scale-up variant calling pipelines on clusters. Because big data scalability requires moving a lot of data between nodes in a big data analytics infrastructure, the current row-based data storage formats and processing row-by-row make these frameworks less efficient for linear scalability and high performance. These solutions use Apache Spark/Hadoop as big data frameworks loosely integrate existing single node pre-processing and variant calling applications. ADAM [22], for example, introduces its own formats, APIs and processing engines. It is built on top of Avro and Parquet for columnar I/O based storage. These solutions come with extra memory overhead and scalability issues.

In order to address these overhead challenges, Apache Arrow in-memory columnar data format in genomics applications has been shown to provide for efficient storage, in-memory analytics and better cache locality exploitation in addition to improved parallel computation [24, 25]. However, limitations in communication overhead remain a challenge, thereby limiting scalability potential of these solutions compared to their custom-made MPI-based HPC alternatives. In this work we establish a case for low-overhead communication using Apache Arrow Flight, enabling efficient scalability of pre-processing and variant calling applications for NGS data on a cluster. This solution leverages the benefits of the Arrow in-memory columnar data format and Arrow Flight wire-speed protocol for shuffling data (between nodes) to sort reads after alignment. The whole workflow is created in Python and Pandas dataframes, which enable computation/analytics like sorting and duplicates removal of NGS data. This solution combines the easy programmability and flexibility of big data pipelines with the high performance and scalability of it HPC-based alternatives. The main contributions of this work are as follows:

- BWA-MEM, a sequence alignment algorithm has been modified to output Arrow in-memory columnar data instead of SAM file. Each BWA-MEM instance on each executor creates 128 Arrow RecordBatches corresponding to chromosomes. This approach stores chromosomes regions level sorted SAM reads.

- Arrow Flight data communication (receiver and transfer protocol) applications have been developed, which communicate with each other to shuffle data through Arrow Flight end-points.
- On each executor node, Arrow data is converted to Pandas dataframes through PyArrow APIs. A Picard MarkDuplicate compatible algorithm for short-reads duplicates removal is developed in Python.
- The whole variant calling pipeline (alignment, sorting, duplicates removal and variant caller) is managed through SLURM workload manager scripts to use in-memory data for intermediate applications.

In summary, this implementation has following advantages over the existing Apache Spark and MPI based workflows:

- As compared to Apache Spark based frameworks, this approach provides more than 2x speedup, better cluster scalability, less memory footprints, efficient system resource utilization and low communication overhead for data shuffling in intermediate applications.
- When comparing with MPI based solutions, this approach has similar performance for runtime but exhibits better cluster scalability. However, Python ease of programmability and simple Arrow Flight based cluster creation through SLURM or with any other workload manager makes this approach more attractive and suitable for people with little knowledge of HPC systems and performance scalability.

This paper is organized as follows. In Section 5.2.2, a brief introduction of the Apache Arrow in-memory data format, the Arrow Flight protocol and the SLURM scheduler is given. Section 5.2.3 outlines some big data based pipelines for NGS data processing. Our implementation for both pre-processing and variant calling is described in Section 5.2.4, followed by Section 5.2.5, where we compare this approach with existing frameworks in both performance and accuracy. In Section 5.2.6, we discussed the results related to speedup, scalability, memory consumption, Arrow Flight throughput. Finally we conclude this work in Section 5.2.7.

5.2.2. Background

In this section, we introduce genome sequencing technologies, NGS data, pre-processing and variant calling followed by a short discussion on Apache Arrow data format, Arrow Flight communication protocol and SLURM workload manager.

Genome sequencing and NGS data: To analyze an organism DNA for the purpose of understanding and characterizing the unique features it exhibits, the proper order of bases of its DNA should be determined. Different sequencing technologies have been invented for this purpose. Previously widely used Sanger sequencing, next generation sequencing (short reads) from Illumina and the latest third generation sequencing technologies (long reads) from PacBio and Oxford Nanopore are most common technologies. These technologies produce massive amounts of raw genome sequencing data. To understand it and extract useful information about the DNA bases variations in a genome, multiple computational processing steps are necessary to clean and arrange this data for down stream analysis.

Pre-processing and variant calling: In comparative genomics, variant calling analysis reveals deep insights into nucleotide-level organismal differences in some specific traits among populations from an individual genome sequence data. To accomplish this analysis NGS data requires a number of pre-processing steps including sequence alignment, chromosome based coordinate sorting, PolymeraseChain Reaction (PCR) duplicates removal and sometimes base quality score re-calibration. These steps are common in all most every variant calling workflow.

Apache Arrow: Apache Arrow is an in-memory standard columnar data format. Due to the columnar data storage, efficient vectorized data analytics operations and better cache locality can be exploited using this format. Apache Arrow [4] is becoming a standard columnar format for in-memory data analytics. Introduced in 2015, Arrow provides cross-language interoperability and IPC by supporting many languages such as C, C++, C#, Go, Java, JavaScript, MATLAB, Python, R, Ruby, and Rust. Arrow also provides support for heterogeneous platforms in the form of rapids.ai for GP-GPUs and Fletcher for FPGA systems [5]. Apache Arrow is increasingly extending its eco-system by supporting different APIs (e.g., Parquet, Plasma Object Store, Arrow Compute, etc.) and many open-source libraries/tools are integrating Arrow inside them for efficient data manipulation and transfer. For example, TensorFlow has recently introduced the TensorFlow I/O [6] module to support the Arrow data format, the Dremio big data framework is built around the Apache Arrow eco-system, pg2arrow (a utility to query PostgreSQL relational database), turbodbc which supports queries in Arrow format, etc.

Arrow stores data in contiguous memory locations to make the most efficient use of CPU cache and vector (SIMD) operations. Moreover, Arrow can efficiently manage big chunks of memory on its own without any interaction with a specific software language run-time, particularly garbage-collected methods. This way, large data sets can be stored outside heaps of virtual machines or interpreters, which are often optimized to work with few short-lived objects, rather than the many large objects used throughout big data processing pipelines. Furthermore, movement or non-functional copies of large data sets across heterogeneous component boundaries are prevented, including changing the form of the data (serialization overhead).

Arrow Flight: Arrow Flight is a submodule in the Apache Arrow project which provides a protocol for transferring bulk Arrow format data across the network. Apache Arrow is also being integrated into Apache Spark for efficient analytics for columnar in-memory data. Arrow Flight [7] provides a high performance, secure, parallel and cross-platform language support (using the Apache Arrow data format) for bulk data transfers particularly for analytics workloads across geograph-

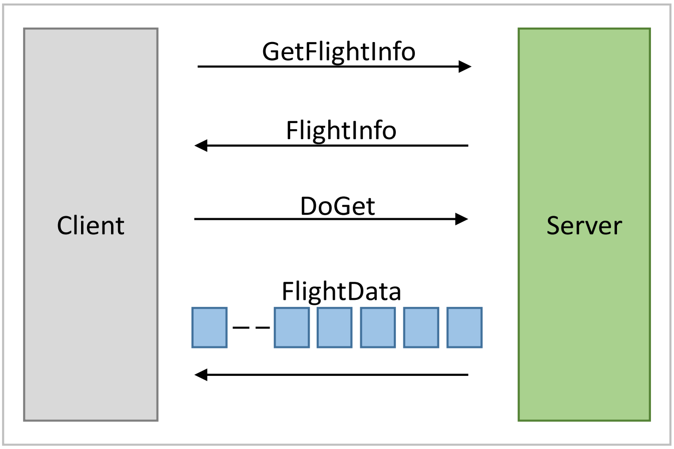


Figure 5.12 A simple Flight setup might consist of a single server to which clients connect and make DoGet requests.

ically distributed networks. Using Apache Arrow as standard data format across all languages/frameworks as well as on the wire allows Arrow Flight data (Arrow RecordBatches) to prevent any serialization/de-serialization when it crosses process boundaries. As Arrow Flight operates directly on Arrow RecordBatches without accessing data of individual rows as compared to traditional database interfaces, it is able to provide high performance bulk operations. Arrow Flight supports encryption out of the box using gRPC built in TLS/OpenSSL capabilities. A simple user/password authentication scheme is provided out-of-the-box in Arrow Flight and provides extensible authentication handlers for some advanced authentication schemes like Kerberos. A simple Arrow Flight client-server setup in which clients connect and establish connection to a server and preform `DoGet()` operations is shown in Figure 5.12. The performance efficiency and throughput of Arrow Flight connection in a remote client-server architecture have been analyzed. The throughput of `DoPut()` (client send a stream of RecordBatches to the server) and the `DoGet()` (client receives a stream back from the server) operations is measured and shown in Figure 5.4. `DoPut()` throughput increasing from 1.9GB/s to 4.5GB/s while `DoGet()` achieves 2.5GB/s to 6GB/s throughput with 1 up to 16 Arrow streams in parallel.

SLURM Scheduler: SLURM is a portable and highly scalable cluster resources management framework. Setting up jobs and resources in SLURM to get bare-metal performance is easy and simple and it also provide both the robustness as well as security needed for HPC applications.

5.2.3. Related Work

In the past two decades, both high performance computing (HPC) programming models (using MPI) and big data frameworks (like Apache Hadoop and Spark) based solutions have been explored rigorously for genomics applications. Many variant calling workflows and tools have been developed over the past decade, including SparkGA2 [23], ADAM [22], SparkBWA [26], BWASpark [27], etc. Similarly, MPI based parallel versions of the BWA aligner have been developed, such as pBWA [28] as well as QUARTIC, the most recent MPI based BWA-MEM (alignment and sorting) [29] algorithm.

5.2.4. Implementation

While Apache Hadoop and Spark based solutions provide a simple and straightforward method for data parallelization for genomics workflows and particularly somatic/germline variant calling pipelines, still the overheads related to data communication, memory usage and better scalability issues for big clusters remain unsolved for such big data frameworks. We combine the benefits of the Apache Arrow columnar in-memory data format in-conjunction with the high performance wire-speed data transfer protocol, Arrow Flight. SLURM's managed private cluster is used for distributed and parallel NGS data processing. In the following, the implementation details of pre-processing (alignment, sorting, duplicates removal) applications and variant calling are discussed.

FASTQ Streaming and BWA-MEM: SeqKit [30] is an efficient multi-threaded command line FASTQ/FASTA data manipulation software. SeqKit runs on a dedicated node and streams out the same number of FASTQ data files as the number of executor nodes available in the cluster in parallel. Each BWA-MEM instance runs on a separate node in a cluster for performance purposes as shown in Figure 5.13. Each BWA-MEM instance produces in total 128 Arrow RecordBatches in ArrowSAM [24] format. For human genomes, this division into separate chromosome chunks is derived from the ability of load-balanced parallel and independent processing of NGS data on multi-cores and multi-nodes computing systems.

Arrow Flight Data Shuffling: Once the distributed BWA-MEM instances finish the alignment task, Arrow Flight sender and receiver applications on each node start sending/receiving a designated number of Arrow RecordBatches from all the connected nodes through an Arrow Flight connection. Each node sends `[128/N]` RecordBatches to all available executor nodes, where `'N'` is the total number of executor nodes connected through Arrow Flight endpoints.

Arrow Data Merging: Flight receiver application on each node collects total of 128 RecordBatches, coming from its own sender and the sender applications of the rest of the executors. All RecordBatches of specific partitions are then merged through pyarrow APIs and converted to pyarrow tables. The resultant tables are efficiently converted to Pandas dataframes for some further sorting and duplicate removal operations/analytics.

Pandas Dataframe Sorting: All dataframes on each node are sorted by coordinates with the `'beginPos'` key in parallel. Sorting on Pandas dataframes with size less than 2GB is efficient.

Pandas Dataframe MarkDuplicate: A Picard "MarkDuplicate" compatible algorithm is developed for duplicate reads removal in pair-end short reads NGS

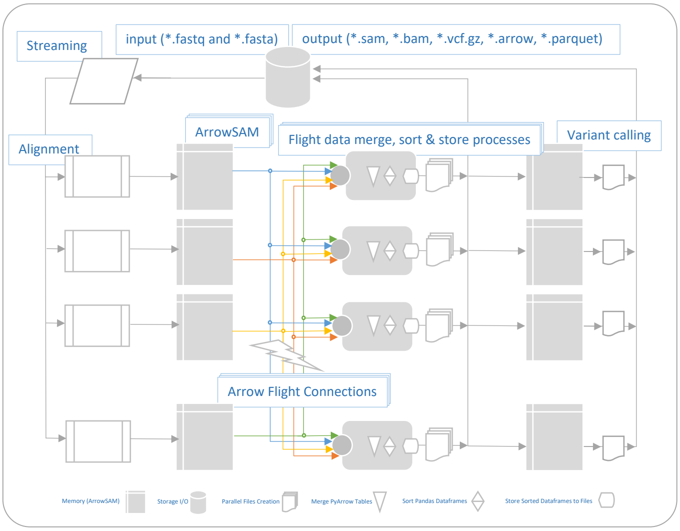


Figure 5.13 Detailed architectural design of pre-processing and variant calling workflow. Input FASTQ data is being streamed to multiple BWA-MEM instances, which create the ArrowSAM output. Arrow RecordBatches are being transferred/received through Arrow Flight. These Arrow RecordBatches are finally merged, sorted, duplicates removed and resultant output is written on IO, followed by variant calling.

data. The sorted dataframes go through this algorithm by updating the ’`Flag`’ field in case the read in a specific reads bundle is detected as a duplicate.

Intermediate Output: For further downstream variant analysis, any variant caller can be selected in this workflow. All mainstream germline (Strelka2, DeepVariant, Octopus) and somatic (Strelka2, Octopus, NeuSomatic) variant callers use (region-specific) chromosome coordinates like `"chr20:10,000,000-10,010,000"`. Since this approach outputs data to the I/O with a total of 128 files, this partition is useful when multiple nodes are used for variant calling. The resultant dataframe(s) can be stored on disk in the conventional BAM file format and/or a columnar output file format options like Arrow, Parquet and compressed Parquet for further downstream analysis. The columnar formats are particularly suited for high performance I/O writing/reading.

Variant Calling: Any variant caller which can support region-specific variant calling can be used in this approach. We use DeepVariant a recent and accurate/fast variant caller as a use case to demonstrate the feasibility of using variant caller in this framework.

5.2.5. Evaluation

This section evaluates the scalability, throughput and speedup we have achieved for pre-processing of NGS sequencing data in alignment, sorting and marking duplicates stages against the existing frameworks. Here we compare two other existing state-of-the-art cluster scaled pre-processing implementations namely, SparkGA2 [23] and QUARTIC [29].

SparkGA2: SparkGA2 [23] is a Apache Spark based cluster scaled implementation of GATK best practices variant calling pipeline. SparkGA2 starts FASTQ streaming application and initiates multiple BWA-MEM instances on Spark executor nodes in parallel. It uses the built-in Scala API in Spark for sorting the aligned reads. As Picard MarkDuplicate algorithm is considered as standard for paired-end reads for duplicates removal, SparkGA2 uses this Picard MarkDuplicate in Spark for distributed processing on cluster.

QUARTIC: QUARTIC (QUick pArallel algoRithms for high-Throughput sequencIng data proCessing) is implemented using MPI. Though this implementation uses I/Os between pre-processing (alignment, sorting and mark duplicate) stages, it still performs better than other Apache Spark based frameworks. These implementations efficiently exploit the multi-cores and multi-nodes parallelization on HPC infrastructure. An MPI wrapper is created for the original BWA-MEM algorithm while using parallel IO and shared memory for alignment. Sorting implements a parallel version of the bitonic sort from scratch in MPI. Their duplicate removal algorithm is based on Picard [31] MarkDuplicate written in MPI.

Experimental Setup

We have performed all the experiments and comparisons on the SurfSara Cartesius [9] HPC cluster (part of the Dutch national supercomputing infrastructure) with each node is a dual socket Intel Xeon server with E5-2680 v4 CPU running at 2.40GHz. Each processor has 14 physical cores with support of 28 hyper-threading jobs. Both processors are connected through Intel QPI (QuickPath Interconnect) and share memory through NUMA (non-uniform memory access) architecture. A total of 192-GBytes of DDR4 DRAM with a maximum of 76.8 GB/s bandwidth is available for whole system. A local storage of 1-TBytes is available on the system. CentOS 7.3 Minimal Server operating system is installed. All nodes are connected through Mellanox ConnectX-3 or Connect-IB InfiniBand adapter.

Dataset: We use Illumina HiSeq generated NA12878 dataset with paired-end reads of WES of human with 30x sequencing coverage. Read length of 100 base-pairs is used for all data. The Human Genome Reference, Build 37 (GRCh37/hg19), is used as a reference genome.

5.2.6. Results

Performance Evaluation

This section evaluates the performance gains in term of runtime speedups, efficiency and cluster scalability as compared to existing frameworks as well as the Arrow Flight throughput over the network which enables the better overall performance of this approach on clusters.

Runtime Speedup: For pre-processing applications, breakdown of execution time of individual applications is shown in Figure 5.14. A scalable trend is ob-

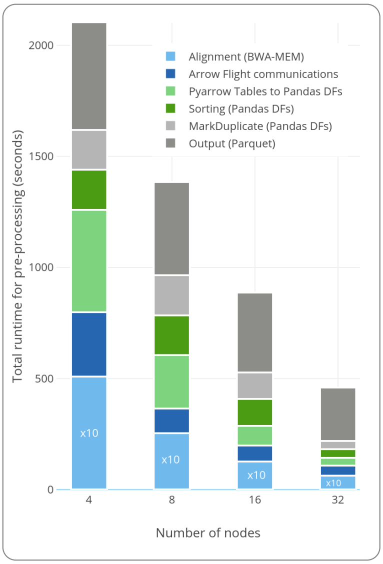


Figure 5.14 Breakdown of execution time for different pre-processing stages on the HG002 dataset. A scalable trend is observed by increasing the number of nodes, also for the communication part which traditionally is a bottleneck for scalability [23].

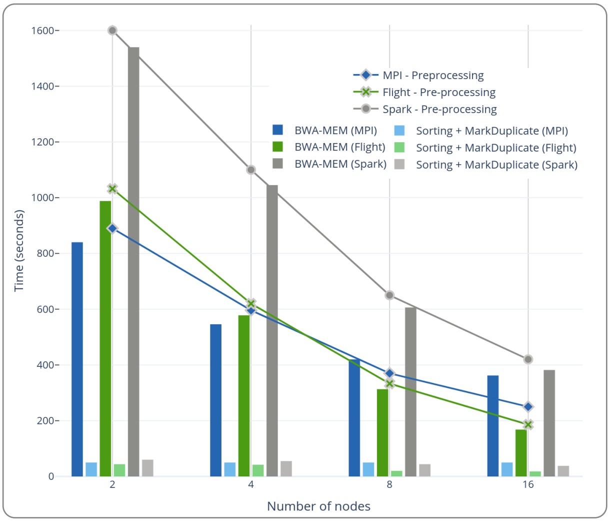


Figure 5.15 Overall pre-processing runtime performance comparison of different approaches by increasing the number of nodes. Apache Spark (SparkGA2), MPI (QUARTIC) and this approach (Apache Arrow and Arrow Flight) are compared.

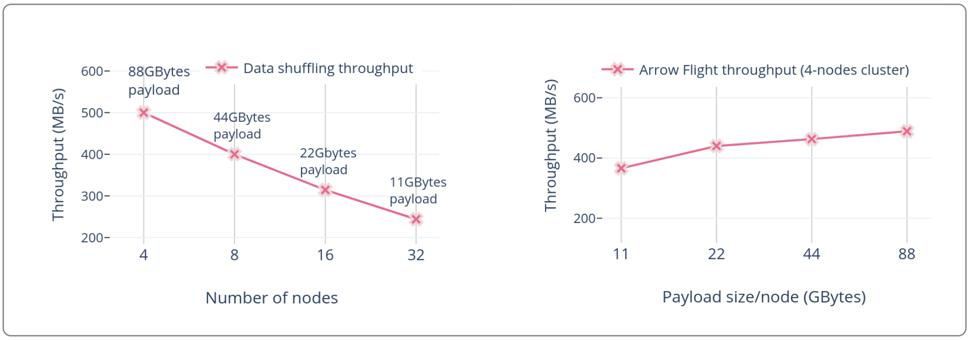


Figure 5.16 Left figure shows the throughput (on each node) in MB/s with different payload sizes (88-11 GB) with varying number of nodes from 4 to 32, while right figure demonstrates the continued increase in throughput (on each node) when increasing input data size from 11 to 88 GB on a 4 node cluster.

served by increasing the number of nodes, also for the communication and data shuffling part in sorting which is explicitly measured here to show a linear decrease in it that is traditionally a bottleneck for scalability on Apache Spark cluster. We also compared the overall runtime of pre-processing applications including BWA-MEM, sorting and duplicates removal with the existing state-of-the-art cluster scaled frameworks. Both the Apache Spark based framework (SparkGA2) and MPI implementation (mpiBWA, mpiSORT and mpiMarkDup) have been run on the same cluster with same datasets. Compared to SparkGA2 pre-processing results, more than 2x and 1.5x speedups are achieved, respectively, for all cluster sizes as shown Figure 5.15. Regarding MPI based comparisons, our approach incurs a marginal overhead for 2 and 4 nodes cluster however increasing the nodes size in cluster to 8 or more nodes, the overall execution time of our approach is decreasing 20%-60%.

Cluster scalability: If the data size remains constant the overall runtime of pre-processing applications can be scaled efficiently by doubling the number of nodes. The total data size also influences the overall performance. As discussed below, Arrow Flight gives better throughput when the Arrow data packet size is big. Normally the Figure 5.15 does not highlight that linear scalability because the total input data is also being being divided by the factor of nodes being increased.

Memory consumption: With the limited memory available per core on clusters, using the Apache Spark framework incurs additional memory overhead due to its built-in Java and Scala codebase which makes it inefficient to process both computation and memory-bound applications. To prevent this extra memory overhead we replace Apache Spark by SLURM, which is a memory-efficient alternative for cluster environments. As shown in Figure 5.13 after BWA-MEM (alignment) application, data shuffling, merging and sorting and finally duplicates removal is solely being done inside memory. Though this requires almost two folds extra memory of total data size but still this is much less than what is used in Apache Spark based implementations.

Table 5.4 Accuracy evaluation of small variants of HG002 (NA24385 with 50x coverage taken from PrecisionFDA challenge V2 datasets) against GIAB HG002 v4.2 benchmarking set. This table shows the SNP and INDEL results for "Chr1" on a single node (default) run.

Variant	Truth total	TP	FN	FP	Recall	Precision	F1-Score
INDEL	42689	42390	299	131	**0.992996**	0.997053	0.995020
SNP	264143	262367	1776	351	**0.993276**	**0.998665**	**0.995963**

Table 5.5 Accuracy evaluation of small variants of HG002 (NA24385 with 50x coverage taken from PrecisionFDA challenge V2 datasets) against GIAB HG002 v4.2 benchmarking set.

Variant	Truth total	TP	FN	FP	Recall	Precision	F1-Score
INDEL	42689	42390	299	127	**0.992996**	**0.997142**	**0.995065**
SNP	264143	262365	1778	355	0.993269	0.998649	0.995952

Arrow Flight Throughput: It has been observed that a maximum of 4.5GB/s throughput is achievable for DoPut() while DoGet() achieves up to 6GB/s throughput with up to 16 streams in parallel on remote hosts as shown in Figure 5.4. But increasing the Flight connections in a cluster also effects this throughput. In this implementation, every Arrow Flight connection on each node communicate (sends and receives) with all available Arrow Flight end-points on all the nodes in a cluster. This Arrow Flights connections scenario creates a network congestion but achieves efficient data shuffling. As shown in Figure 5.16, a maximum 500 MB/s throughput was achievable in a 4 nodes cluster when each node is sending more than total regions files (128) / number of nodes (4) in each iteration to its neighboring nodes at the same time. This also shows that with increasing the Arrow data packet size in each Arrow Flight steams promises much better throughput on even small cluster.

Accuracy

SNP and INDEL variants detection accuracy of DeepVariant variant caller has been compared in a single node and distributed environment. We used HG002 (NA24385 sample with 50x coverage taken from PrecisionFDA challenge V2) dataset to detect SNP and INDEL variants using DeepVariant (v1.1.0), against GIAB v4.2 benchmark set for HG002 dataset. The GA4GH small variant benchmarking tool hap.py [32] has been used to compare the resulting variants in both methods. Table 5.4 and Table 5.5 list the accuracy analysis results in terms of recall, precision and F1-score for the single node and distributed approach, respectively.

5.2.7. Conclusion

This work demonstrates the efficient usage of the Apache Arrow data format and Arrow Flight communication protocol to ensure low-latency communication of genomics data in a cluster environment. Arrow Flight allows for effective scalability of genomics pipelines on large clusters, while eliminating communication time as a scalability bottleneck.

Almost all existing frameworks for processing genomics data are built around big data frameworks like Apache Hadoop and Apache Spark, which does not benefit from columnar in-memory data processing on vector units nor exploit the caches locality efficiently. These frameworks also cost extra memory overheads. Our solution uses the SLURM workload manager as an application handler and data scheduler to replace Apache Spark framework or MPI based implementation of genomics applications. Our approach allows to process more columnar data in-memory without worrying about the extra memory costs. Using SeqKit to create chunks and streaming the resultant FASTQ input to BWA-MEM instances eliminates the additional processing time. We have shown that BWA-MEM is being scaled almost linearly while initiating only one instance per node. Through this approach we are also able to achieve 1.5x and 2x speedup over existing state of the art frameworks like SparkGA2. The performance comparisons of this approach with MPI based implementation gives similar run-times with better clusters scalability and applications flexibility. Integrating Arrow Flight microservices into existing data transfer and analytics frameworks (Apache Spark, TensorFlow, XGBoost, etc.,) for distributed and scalable processing exhibits both parallel and high throughput data transfer and compute capabilities. Also Arrow Flight based distributed Apache Arrow data scheduling, compute and query services like DataFusion and Ballista present applications for this purpose.

REFERENCES

[1] T. Li, M. Butrovich, A. Ngom, W. S. Lim, W. McKinney, and A. Pavlo. "Mainlining Databases: Supporting Fast Transactional Workloads on Universal Columnar Data File Formats". In: *Proc. VLDB Endow.* 14.4 (Dec. 2020), pp. 534–546. issn: 2150-8097. doi: 10.14778/3436905.3436913. url: https://doi.org/10.14778/3436905.3436913.

[2] F. Färber, S. K. Cha, J. Primsch, C. Bornhövd, S. Sigg, and W. Lehner. "SAP HANA Database: Data Management for Modern Business Applications". In: *SIGMOD Rec.* 40.4 (Jan. 2012), pp. 45–51. issn: 0163-5808. doi: 10.1145/2094114.2094126. url: https://doi.org/10.1145/2094114.2094126.

[3] D. Huang, Q. Liu, Q. Cui, Z. Fang, X. Ma, F. Xu, L. Shen, L. Tang, Y. Zhou, M. Huang, W. Wei, C. Liu, J. Zhang, J. Li, X. Wu, L. Song, R. Sun, S. Yu, L. Zhao, N. Cameron, L. Pei, and X. Tang. "TiDB: A Raft-Based HTAP Database". In: *Proc. VLDB Endow.* 13.12 (Aug. 2020), pp. 3072–3084. issn: 2150-8097. doi: 10.14778/3415478.3415535. url: https://doi.org/10.14778/3415478.3415535.

[4] Apache. *Apache Arrow: A cross-language development platform for in-memory data [Accessed 29th Dec. 2019]*. 2019. url: https://arrow.apache.org/.

[5] J. Peltenburg, J. van Straten, M. Brobbel, H. P. Hofstee, and Z. Al-Ars. "Supporting Columnar In-memory Formats on FPGA: The Hardware Design of Fletcher for Apache Arrow". In: *Applied Reconfigurable Computing*. Ed. by C. Hochberger, B. Nelson, A. Koch, R. Woods, and P. Diniz. Cham: Springer International Publishing, 2019, pp. 32–47. isbn: 978-3-030-17227-5.

[6] tensorflow. *Dataset, streaming, and file system extensions maintained by TensorFlow SIG-IO*. 2021. url: https://github.com/tensorflow/io.

[7] A. Arrow. *Arrow Flight RPC [Accessed 29th July. 2021]*. 2019. url: https://arrow.apache.org/docs/format/Flight.html.

[8] M. Raasveldt and H. Mühleisen. "Don't Hold My Data Hostage: A Case for Client Protocol Redesign". In: *Proc. VLDB Endow.* 10.10 (June 2017), pp. 1022–1033. issn: 2150-8097. doi: 10.14778/3115404.3115408. url: https://doi.org/10.14778/3115404.3115408.

[9] SurfSara. *Cartesius: the Dutch supercomputer*. 2020. url: https://userinfo.surfsara.nl/systems/cartesius.

[10] iperf3. *iperf3: A TCP, UDP, and SCTP network bandwidth measurement tool*. 2020. url: https://github.com/esnet/iperf.

[11] Mellanox. *InfiniBand write bandwidth tool*. 2021. url: https://community.mellanox.com/s/article/ib-write-bw.

[12] R. Murray. *Spark source for Flight enabled endpoints*. 2021. url: https://github.com/rymurr/flight-spark-source.

[13] B. Cutler. *Apache Arrow Flight service with Apache Spark and TensorFlow clients*. 2021. url: https://github.com/BryanCutler/SparkArrowFlight.

[14] ehenry. *A high performance microservice framework for XGBoost models*. 2021. url: https://github.com/ehenry2/xgbatch.

[15] DataRobot. *Scoring Data*. 2021. url: https://www.datarobot.com/wiki/scoring/.

[16] P. Tholoniat. *A tiny PyGrid node on steroids with Arrow Flights*. 2021. url: https://github.com/tholop/FlightGrid.

[17] IBM. *The Mesh for Data platform - Arrow/Flight module*. 2021. url: https://github.com/mesh-for-data/arrow-flight-module.

[18] A. Arrow. *Proposal for Flight SQL extension on top of FlightRPC [Accessed 29th August. 2021]*. 2021. url: https://issues.apache.org/jira/browse/ARROW-9825.

[19] J. M. Zook and et al. “Extensive sequencing of seven human genomes to characterize benchmark reference materials”. In: *Scientific Data* 3.1 (June 2016), p. 160025. issn: 2052-4463. doi: 10.1038/sdata.2016.25. url: https://doi.org/10.1038/sdata.2016.25.

[20] e. a. Kline. “Targeted next-generation sequencing of pediatric neuro-oncology patients improves diagnosis, identifies pathogenic germline mutations, and directs targeted therapy”. In: *Neuro-Oncology* 19.5 (Nov. 2016), pp. 699–709. issn: 1522-8517. doi: 10.1093/neuonc/now254. eprint: https://academic.oup.com/neuro-oncology/article-pdf/19/5/699/17687132/now254.pdf. url: https://doi.org/10.1093/neuonc/now254.

[21] D. Decap, J. Reumers, C. Herzeel, P. Costanza, and J. Fostier. “Halvade: scalable sequence analysis with MapReduce”. eng. In: *Bioinformatics (Oxford, England)* 31.15 (Aug. 2015). btv179[PII], pp. 2482–2488. issn: 1367-4811. doi: 10.1093/bioinformatics/btv179. url: https://doi.org/10.1093/bioinformatics/btv179.

[22] M. Massie, F. Nothaft, C. Hartl, C. Kozanitis, A. Schumacher, A. D. Joseph, and D. A. Patterson. *ADAM: Genomics Formats and Processing Patterns for Cloud Scale Computing*. Tech. rep. UCB/EECS-2013-207, EECS Department, University of California, Berkeley, 2013.

[23] H. Mushtaq, F. Liu, C. Costa, G. Liu, P. Hofstee, and Z. Al-Ars. “SparkGA: A Spark Framework for Cost Effective, Fast and Accurate DNA Analysis at Scale”. In: *Proceedings of the 8th ACM International Conference on Bioinformatics, Computational Biology,and Health Informatics*. ACM-BCB ’17. Boston, Massachusetts, USA: ACM, 2017, pp. 148–157. isbn: 978-1-4503-4722-8. doi: 10.1145/3107411.3107438. url: http://doi.acm.org/10.1145/3107411.3107438.

[24] T. Ahmad, J. Peltenburg, N. Ahmed, and Z. Al-Ars. “ArrowSAM: In-Memory Genomics Data Processing through Apache Arrow Framework”. In: (2019). doi: 10.1101/741843.

[25] T. Ahmad, N. Ahmed, Z. Al-Ars, and H. P. Hofstee. “Optimizing performance of GATK workflows using Apache Arrow In-Memory data framework”. In: *BMC Genomics* 21.10 (Nov. 2020), p. 683. issn: 1471-2164. doi: 10.1186/s12864-020-07013-y. url: https://doi.org/10.1186/s12864-020-07013-y.

[26] J. M. Abuín, J. C. Pichel, T. F. Pena, and J. Amigo. “SparkBWA: Speeding Up the Alignment of High-Throughput DNA Sequencing Data”. In: *PLOS ONE* 11.5 (May 2016), pp. 1–21. doi: 10.1371/journal.pone.0155461. url: https://doi.org/10.1371/journal.pone.0155461.

[27] B. Institute. *BWA on Spark*. 2018. url: https://gatk.broadinstitute.org/hc/en-us/articles/360037225092-BwaSpark-BETA-.

[28] D. Peters, X. Luo, K. Qiu, and P. Liang. *Speeding Up Large-Scale Next Generation Sequencing Data Analysis with pBWA*. 2017. url: https://www.scitechnol.com/JABCB/JABCB-1-101.pdf.

[29] F. Jarlier, N. Joly, N. Fedy, T. Magalhaes, L. Sirotti, P. Paganiban, F. Martin, M. McManus, and P. Hupé. “QUARTIC: QUick pArallel algoRithms for high-Throughput sequencIng data proCessing [version 3; peer review: 2 approved]”. In: *F1000Research* 9.240 (2020). doi: 10.12688/f1000research.22954.3.

[30] W. Shen, S. Le, Y. Li, and F. Hu. “SeqKit: A Cross-Platform and Ultrafast Toolkit for FASTA/Q File Manipulation”. In: *PLOS ONE* 11.10 (Oct. 2016), pp. 1–10. doi: 10.1371/journal.pone.0163962. url: https://doi.org/10.1371/journal.pone.0163962.

[31] *Picard toolkit*. http://broadinstitute.github.io/picard/. 2019.

[32] P. Krusche. *Haplotype VCF comparison tools*. 2021. url: https://github.com/Illumina/hap.py.

6

MPI FOR SCALABILITY OF ALIGNERS AND VARIANT CALLING

Prelude: *In the previous chapters, we have extensively explored the performance optimization paradigms for genome variant calling workflows using Apache Arrow for both single node and cluster scalability in big data frameworks like Apache Spark. In this chapter, we adopt an orthogonal approach (GenMPI) of using parallel programming model like MPI to investigate the performance and scalability of the BWA-MEM and Minimap2 aligners for short and long reads, respectively, as well as the complete variant calling pipeline. GenMPI ensures the exact same accuracy of variants as compared to the original single node methods. In contrast, other big data based solutions cause a slight degradation in variant accuracy in order to achieve better scalability. For the pre-processing steps like sorting and indexing we use Samtools while Sambamba is used for duplicate reads removal. For pair-end short-reads (Illumina) data, we integrate the BWA-MEM aligner and three variant callers (GATK HaplotypeCaller, DeepVariant and Octopus), while for long-reads data, we integrate the Minimap2 aligner and three different variant callers (DeepVariant, DeepVariant with WhatsHap for phasing (PacBio) and Clair3 (ONT)). The contents of this chapter are based on our article entitled "MPI-based Cluster Scalable Variant Calling for Short/Long Reads Sequencing Data" which has been submitted for publication.*

6.1. Overview

Rapid technological advancements in sequencing technologies allow producing cost effective and high volume sequencing data. Processing this data for real-time clinical diagnosis is potentially time-consuming if done on a single computing node. This work presents a complete variant calling workflow, implemented using the Message Passing Interface (MPI) to leverage the benefits of high bandwidth interconnects. This solution (GenMPI) is portable and flexible, meaning it can be deployed to any private or public cluster/cloud infrastructure. GenMPI ensures the exact same accuracy of variants as compared to the original single node methods. In contrast, other big data based solutions result in a slight degradation in variant accuracy in order to achieve better scalability. GenMPI allows using any alignment or variant calling application with minimal adaptation. To achieve high performance, compressed input data can be streamed in parallel to alignment applications while uncompressed data can use internal file seek functionality to eliminate the bottleneck of streaming input data from a separate node. Alignment output can be directly stored in multiple chromosome-specific SAM files or a single SAM file. After alignment, a distributed queue using MPI RMA (Remote Memory Access) atomic operations is created for sorting, indexing, marking of duplicates (if necessary) and variant calling applications. We also show that for 300x coverage data, alignment scales almost linearly up to 64 nodes (8192 CPU cores). Overall, this work outperforms existing big data based workflows by a factor of two and is almost 20% faster than other MPI-based implementations for alignment without any extra memory overheads. Sorting, indexing, duplicate removal and variant calling is also scalable up to 8 nodes cluster. For pair-end short-reads (Illumina) data, we integrated the BWA-MEM aligner and three variant callers (GATK HaplotypeCaller, DeepVariant and Octopus), while for long-reads data, we integrated the Minimap2 aligner and three different variant callers (DeepVariant, DeepVariant with WhatsHap for phasing (PacBio) and Clair3 (ONT)). All codes and scripts are available at:
https://github.com/abs-tudelft/gen-mpi

6.2. Introduction

Next Generation Sequencing (NGS) technologies can produce high throughput, less erroneous and higher depth/coverage sequencing data at low costs. These high-throughput sequencing technologies are making their way from research to the field in a wide range of applications ranging from clinical diagnostics to agriculture. Depending on the experiment design type, the need of sequencing coverage varies. A typical 300x coverage human genome dataset size exceeds 2.3 TBytes [1]. Processing such large datasets on a single-node can take up to several days. For a number of applications employing higher coverage sequencing data and/or requiring urgent sequencing results, this exceeds acceptable time constraints.

Sequencing coverage influences both accuracy and the sensitivity of genomics analysis. In pediatric brain-tumor studies [2], more than 200x coverage for the tumor sample and more than 100x coverage for the normal sample are collected for better precision for the tumors of concern. Recent studies show that whole exome sequencing (WES)/whole genome sequencing (WGS) helps with diagnosis, decision making, and treatment of fetal diseases [3]. Although WES is normally used to detect fetal anomalies in prenatal and perinatal testing, which only targets protein-coding regions of genes in a genome, this type of sequencing needs urgent and fast sequencing analysis due to the time-critical nature of these tests. Sequencing can also enable finding some rare hereditary disorders and genetic variants associated with specific diseases in newborn screening. A recent study that uses genomic sequencing for newborn screening [4] showed that some of enrolled healthy newborns and children with metabolic diseases or hearing loss exhibited pathogenic variants associated with hereditary breast or ovarian cancer and a pathogenic variant in the gene associated with Lowe syndrome. This shows that sequencing in newborn screening can play a vital role in timely diagnosis and treatment of diseases and ultimately will lead to urgent need of processing genomics data in such time-critical clinical settings.

Many consortia, associations and government disease diagnosis and drug regulatory agencies stipulate guidelines/protocols for genomics sequencing as a standard tool in diagnosis and treatment of some fatal diseases [5]. Many medical and diagnosis centers, particularly in developed countries, have started sequencing as a regular practice for prenatal and perinatal testing, newborn screening, genetic and cancerous disease diagnostic and personalized treatments. In the coming decades, as sequencing becomes a normal practice for human health and other types of research, locally available compute infrastructure to any organization will not be adequate to fulfill the sequencing requirements. At the same time, large computing infrastructure also requires human resources and incurs power and maintenance costs.

Mostly, genome sequencing machines produce FASTQ format raw data for a given sample [6]. Mapping this raw data to a reference genome, sorting the resultant Sequence Alignment/Map (SAM) reads according to chromosomes and their positions, and finally removing the duplicate reads are some of the standard practices before actual variant detection. A large number of tools is available for the aforementioned methods but almost all are developed for use in a single compute node context. Scaling these tools for cluster environments for both scalability and reproducibility is an ongoing challenge. In the past decade, many cluster-scaled solutions have been created. Almost all these solutions use big data frameworks like Apache Spark [7] or Apache Hadoop [8, 9] for distributing and work scheduling. Data formats like Apache Parquet, Apache Arrow, Apache Avro have been explored extensively in conjunction with these frameworks to store and process genomic data efficiently. These frameworks include ADAM [10], SparkGA2 [11], VC@Scale [12] and Halvade [13]. Due to many underlying dependencies, inefficient memory usage, issues related to scalability, cluster deployment challenges as well as incompatible data formats, solutions based on these frameworks are still not widely used in the mainstream Bioinformatics community. While the Message Passing Interface (MPI) has been used previously for parallelization of older genomic algorithms on HPC systems like pBWA [14], in this work we propose a comprehensive approach towards using MPI for cluster scale parallelization of state-of-the-art genomics applications and complete variant calling workflows.

We expect to benefit from the following advantages that MPI has over traditional big data frameworks for genomics workflows.

- Bare-metal performance and linear scalability of existing applications;
- Portable access to low-level network capabilities;
- Little to no extra memory overheads otherwise incurred in big data frameworks;
- Efficient MPI I/O performance on parallel file systems (like Lustre [15], GPFS [16]).

In the following, we list the main contributions of this work.

- MPI-based parallelization of BWA-MEM [17] and Minimap2 [18], compressed input FASTQ files can be provided as separate files to parallelize streaming of input while reading of uncompressed FASTQ will be parallelized internally. SAM output is tested on both POSIX and shared MPI I/O.
- Sorting, indexing and duplicate reads removal (if necessary) can be performed through a queue employing low-level network atomic operations (if input is already chunked based on chromosomes) or through MPI based bitonic sorting.
- For short reads, GATK HaplotypeCaller [19], Octopus [20], and DeepVariant [21] are used in combination with the MPI RMA-based queue for parallel chromosomes processing on a cluster.
- For long reads, DeepVariant, DeepVariant with WhatsHap [22] for phasing, and Clair3 [23] variant callers are used in combination with the MPI RMA-based queue for parallel chromosome processing on a cluster.
- Resultant VCFs are merged through Bcftools [24] to generate a single combined VCF (variant calling file) and to insure variants correctness.
- SNP/INDEL accuracy/precision/F1 tests are performed through hap.py [25] against GIAB v4.2.1 [26] benchmark set for the HG002 dataset.

The rest of this article is organized as follows. In Section 6.3 we describe our implemented methods in detail for both pre-processing and variant calling for short and long reads NGS data, followed by Section 6.4 where we compare the methods integrated into this approach with the existing workflows for both performance, accuracy, and scalability. In Section 6.5, run-time, accuracy, scalability, portability, and cost efficiency are discussed briefly. Finally we conclude this work in Section 6.6.

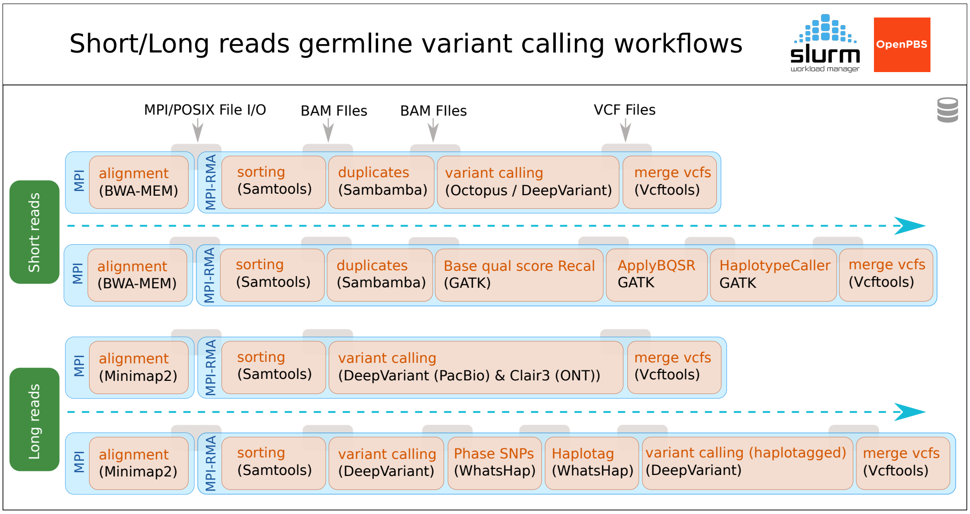


Figure 6.1 Architectural description of short and long reads based NGS data variant calling workflows.

6.3. Methods

We have constructed both short and long reads based cluster scalable variant calling workflows. For pre-processing of short and long reads NGS data, the *BWA-MEM* [17] aligner with sorting (*Samtools* [27]) and mark duplicate (*Sambamba* [28]) are used for the former, while *Minimap2* [18] with sorting (*Samtools*) is used for the latter. As shown in Figure 6.1, for short reads we use three different variant callers like *Octopus* and *DeepVariant* as well as GATK best practices variant calling pipeline using *HaplotypeCaller*. Similarly, for PacBio long reads, DeepVariant and DeepVariant with chromosomes phasing using *WhatsHap* have been used while *Clair3* variant caller is used for Oxford Nanopore Technologies (ONT) data as recommended by ONT [29]. These workflows can be run through both cluster workload managers like Slurm [30] and PBS [31]. The following sections provide a more detailed description of both short and long reads workflows implementation.

6.3.1. Short-reads variant calling workflow

In the following subsections, we describe in details how MPI has been integrated with the BWA-MEM short read aligner and how different pre-processing applications are used in the variant calling workflow.

Alignment: The complete algorithmic implementation details of integrating the alignment applications is given in pseudo-code representation in Algorithms 1 and 2. Normally, pair-end short reads are used for variant detection, consisting of two FASTQ raw NGS data files. As shown in Algorithm 1, we start MPI after

Algorithm 1: Part-1: MPI integration into **BWA-MEM** and **Minimap2** for reading compressed and uncompressed FASTQ input files.

```
   // BWA-MEM/Minimap2 basic initialization
1  MPI_Init(NULL, NULL)                                                  ▷ MPI initialization
2  $P \leftarrow$ size (total number of processes);
3  $N \leftarrow$ rank (process number);
4                     ▷ Check if user needs a single SAM or chromosomes-based SAM file(s) to write output
5  if $chromosomes \leftarrow$ output then
6      for $CHRM \leftarrow 1$ to chromosomes do
7          files[CHRM] $\leftarrow$ fopen($CHRM$);
8      end for
9  else
10     file $\leftarrow$ fopen(outputfile)
11 end if
12                                  ▷ Compressed or uncompressed input FASTQ file(s) check
13 if input(compressed (*.gz/*.tar.gz/*.zip)) then
       // FASTQ streaming from an external process
14     Seekable $\leftarrow$ $FALSE$;
15     $parts[n] \leftarrow$ scandir($./parts$);
16     $FASTQ\_INPUT \leftarrow parts[P]$;
17 else
       // SEEK_SET pointer initialization
18     Seekable $\leftarrow$ $TRUE$;
19     size (bytes) $\leftarrow$ Calculate file(s) size;
20     SEEK pointer* $\leftarrow$ size (bytes)/P;
21     gzseek(file, SEEK pointer*);
22 end if
```

basic parameter initialization. First, we check (Lines 5–10) if the user enabled chromosomes-based output into separate SAM files. By default this option is disabled and only single output SAM file generation is enabled. For compressed FASTQ input files (Lines 13–16), each BWA-MEM process expects equally chunked FASTQ pairs in a directory "parts" in the path of the original FASTQ files. This is done through an extra FASTQ streaming process. For streaming purpose, the user has to start an additional process; we used SeqKit [32] for this purpose. SeqKit is an efficient multi-threaded command line FASTQ/FASTA data manipulation tool. In the case of uncompressed FASTQ files (Lines 18–21), we used gzseek() function which sets the file position to a given offset. We get this offset by dividing the total FASTQ file size to the number of total MPI processes. Through this technique the kseq_read() function may encounter a broken first read, which we simply discard because process $N-1$ will read the last read even if it reaches a break point for total number of bytes it can read, as shown in Algorithm 2 (Lines 1–8) in *process()* function *step-0*. In this way, we ensure that none of the input reads is dropped by any process. We calculate the size of each read and increment the bytes counter until it reaches the required number of bytes for each BWA-MEM process. Afterwards, BWA-MEM starts processing these sequences (Lines 9) in *process()* function *step-1*. Finally, we distinguish the chromosomes id for each read if writing to individual chromosome files is enabled (Lines 10–17) in *process()* function *step-2*. The output files should be written on a parallel file system with either POSIX or shared MPI I/O, depending on the best possible performance scenarios for the user.

MPI RMA-based chrosome queue: The MPI RMA interface provides applications with access to advanced features of modern high-performance networks, including direct access to remote memory through puts, gets, and atomic operations [33]. We utilize the atomic fetch-and-add functionality of MPI RMA to assign chromosomes to processes. Processes continuously increment the chromosome

Algorithm 2: Part-2: MPI integration into **BWA-MEM** and **Minimap2** for processing FASTQ data and saving output to SAM single or chromosome-based file.

```
while ($SEQs \neq \emptyset$) & (FASTQ(compressed_file_chunk) ≠ END || FASTQ(SEEK_SET) ≠ END) do
    $SEQs$ ← bseq_read($FASTQ\_INPUT$) // process() -> Step:0
    if Seekable = $TRUE$ then
        bytes+=Bytes count for each read (seq, comment and qual length);
        if bytes ≥ max(SEEK_SET) then
            break;
        end if
    end if
    $SAM$ ← mem_process_seqs($SEQs$)// process() -> Step:1
    if $chromosomes$ ← output // process() -> Step:2
     then
        for $CHRM$ ← 1 to chromosomes do
            files[CHRM] ← err_fputs($files[CHRM], SAMs$);
        end for
    else
        file ← err_fputs($file, SAMs$);
    end if
end while
                                                        ▹ Close the output SAM file(s)
if $chromosomes$ ← output then
    for $CHRM$ ← 1 to chromosomes do
        files[CHRM] ← fclose($CHRM$);
    end for
else
    file ← fclose(outputfile);
end if
MPI_Finalize()                                          ▹ MPI finalization
return $0$;
```

counter atomically until the returned value is equal to or larger than the number of chromosomes. Algorithm 3 shows the algorithm for allocating a suitable window and then querying chromosome numbers and processing them until all chromosomes have been assigned to a process. The respective window is allocated using the "`osc_rdma_acc_single_intrinsic`" info key set to true (Lines 1–2), which is supported by Open MPI and allows the implementation to utilize low-level hardware atomic operations by signaling the use of a single data element [34]. Processes continuously increment a variable using an atomic fetch-and-op in the window to acquire the next chromosome to process (Lines 5–7) and stop once the counter exceeds the number of chromosomes available (Line 9).

Algorithm 3: MPI RMA atomic operations for creating a chromosome queue for pre-procssing and variant calling

```
    Input: NumChromosomes – Number of chromosomes
 1  info ← MPI_Info_create();
 2  MPI_Info_set(info, "osc_rdma_acc_single_intrinsic", "true")  win ← MPI_Win_allocate(sizeof(int), 1, info)  $X$ ← 0
 3  $one$ ← 1
 4  while true do
 5      $X$ ← MPI_Fetch_and_op(&one, MPI_SUM, win)                   ▹ Next Chromosome
 6      if $X$ < NumChromosomes then
 7          process_chromosome($X$)      ▹ Process Chromosome (sort, index, duplicates, variant call)
 8      else
 9          break                              ▹ All chromosomes have been assigned, stop.
10      end if
11  end while
12  MPI_Win_free(win)                                             ▹ Free window resources
```

Sorting, duplicates removal, indexing and variant calling: As described above, the MPI atomic operations based queue algorithm continuously pools the input chromosomes SAM files and operates on them for sorting, mark duplicate, index generation and variant calling algorithms. We used and integrated the sorting algorithm from Samtools and the mark duplicate algorithm from Sambamba because both perform better and have multi-threading capabilities. For variant calling, we use three different tools (GATK HaplotypeCaller, DeepVariant and Octopus) for germline variant calling of short reads. As discussed above, generating SAM files for individual chromosomes in the alignment process is not necessary, since we can also use a single SAM file. If the user wants to use all other pre-processing (sorting, mark duplicate, BAM indexing) and variant calling tools in this workflow, it is also possible to parallelize these by providing the contigs/chromosome numbers, even in the case of using a single SAM file. Finally, the individual VCFs created are merged through Bcftools to produce a final complete VCF file for further downstream analysis.

6.3.2. Long-reads variant calling workflow

In the following subsections, we describe in detail how MPI has been integrated with the long read aligner Minimap2, and how different pre-processing applications are used in the variant calling workflow.

Alignment: Circular consensus sequencing (CCS) based PacBio HiFi reads and Oxford Nanopore technologies (ONT) long reads are making their way from research to clinical applications. They provide more in-depth and better consensus particularly in more complex repetitive regions of the genome [35]. The Minimap2 long reads aligner (with or without some additional parameter settings) is mainly being used for both PacBio and ONT long reads data. The complete algorithmic implementation details of this integration are given in pseudo-code in Algorithms 1 and 2, the detail description of Minimap2 implementation is similar to BWA-MEM as described in Section 6.3.1.

Sorting, indexing and variant calling: For long reads after alignment, only sorting and index generation for BAM is necessary. MPI atomic operations based queue algorithm continuously polls the input chromosomes SAM files, or otherwise contigs/chromosomes number can also be used in case a single SAM file is generated through Minimap2. Then the queue algorithm operates on those files/contigs/chromosomes for sorting, BAM index generation and variant calling algorithms. We used and integrated the sorting algorithm from Samtools. For variant calling, we used three algorithms for germline variant calling of long reads (DeepVariant, DeepVariant with WhatsHap for phasing on PacBio dataset, and Clair3 for ONT dataset). These variant callers are commonly recommended by the corresponding sequencing vendors. Finally, the individual VCFs created are merged using Bcftools to produce a final complete VCF file for further downstream analysis.

6.4. Results

Evaluation cluster: We used both HLRS Hawk [36] (an HPE Apollo 9000 system at the High Performance Computing Center Stuttgart (HLRS) in Germany) and the SurfSara Snellius [37] (part of the Dutch national supercomputing infrastructure)

HPC clusters. Each compute node of Hawk is equipped with a dual socket AMD EPYC 7742 processors (64 cores/socket) running at 2.25 GHz. All nodes are connected through Mellanox HDR200 (interconnect bandwidth 200 Gbit/s) Infiniband adapter. Likewise, on SurfSara Snellius, each compute node is equipped with a dual socket AMD EPYC 7H12 (64 cores/socket) processors running at 2.6 GHz. All nodes are connected through Mellanox HDR100 (interconnect bandwidth 100 Gbit/s) Infiniband adapter. A local storage of 1-TBytes and the same amount of network attached storage is available on both systems.

The parallel file system available on HLRS Hawk is based on Lustre while SurfSara Snellius is equipped with IBM Spectrum Scale (GPFS) [16]. The SLURM Workload Manager is installed on SurfSara Snellius while HLRS Hawk uses PBSPro workload manager and job scheduler.

Datasets: We used the Illumina, PacBio HiFi, and ONT HG002 datasets taken from the PrecisionFDA challenge V2 [38]) dataset for variant calling workflows. We also used 300x sequencing coverage WGS data from Genome in a Bottle (GIAB) aligned with novoalign for the Illumina HiSeq 300x reads for NA12878 [39] to analyze the scalability of BWA-MEM aligners. Human genome reference GRCh38 [40] is used as a reference genome. For accuracy comparisons, the GIAB v4.2.1 [26] benchmark set for HG002 dataset is used.

6.4.1. Runtime performance

In this section, we analyze and compare the runtime performance of MPI based aligners (BWA-MEM and Minimap2) with existing state-of-the-art MPI and Apache Spark based implementations. We also benchmark the performance of variant calling workflows using different variant callers on a cluster of up to 8 nodes.

Short reads alignment: We compare the scalability and runtime performance of GenMPI with those based on Apache Spark (ADAM's Cannoli [10] alignment) and another MPI-based implementation, QUARTIC [41] (mpiBWA). We use different numbers of nodes; 2, 4, 8 and 16 nodes for runtime performance comparisons. ADAM's Cannoli uses the built-in Scala API from the Apache Spark backend for distributing and scheduling data for parallel processing. The Cannoli wrapper encompasses many different aligners and variant callers for distributed processing. Apache Spark based implementations use HDFS or NFS for I/O operations. QUARTIC (mpiBWA) is a distributed BWA-MEM alignment algorithm employing MPI functionality and uses MPI shared I/O for input/output on parallel file system. In Figure 6.2 (left bar), we show the BWA-MEM runtime on a single node utilizing all 128 system CPU cores with one thread for each core. We consider this time as an ideal runtime for a single node. When we compare this runtime by increasing the number of nodes in a cluster as shown in Figure 6.2, we see QUARTIC and GenMPI perform better than this ideal runtime, while ADAM Cannoli BWA-MEM implementation has less than ideal scalability. We attribute the super-linear scaling of both QUARTIC and GenMPI to scalability issues of BWA-MEM when utilizing all cores on a single node using a single process (128 threads). As a consequence, we use two MPI processes on each node to overcome this inefficiency for all other runs. Overall GenMPI outperforms QUARTIC by almost 20% in terms

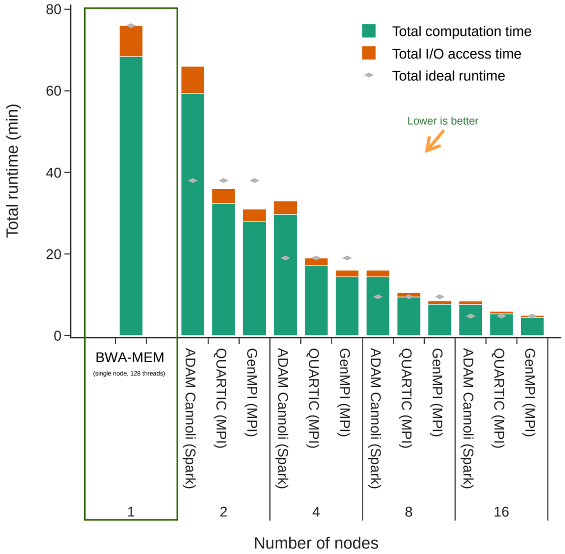
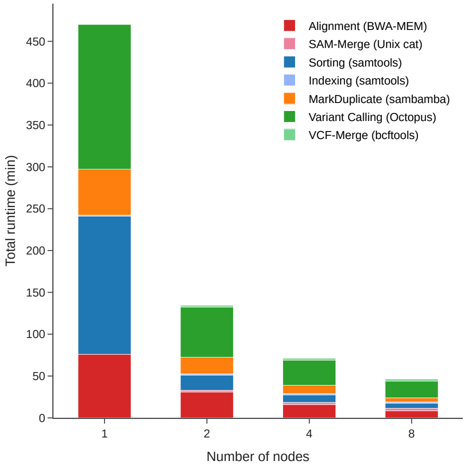


Figure 6.2 Run-time and scalability comparisons of ADAM's Cannoli, QUARTIC mpiBWA and this work implementations of BWA-MEM aligner on a cluster of varying number of nodes using HG002 (NA24385) Illumina NovaSeq 35x coverage dataset. Ideal runtime is the runtime assuming ideal scalability on the available nodes.



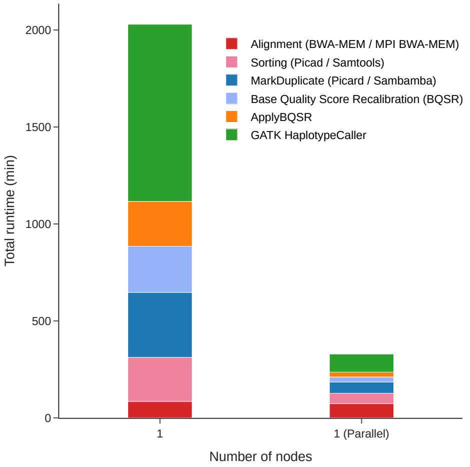


Figure 6.3 Run-time and scalability benchmark of GenMPI for BWA-MEM aligner and Octopus variant caller on a cluster of varying number of nodes using HG002 (NA24385) Illumina NovaSeq 35x coverage dataset.

Figure 6.4 Run-time and scalability benchmark of GenMPI for BWA-MEM aligner and GATK best practices pipeline on a single node versus alternative parallel single node methods using HG002 (NA24385) Illumina NovaSeq 35x coverage dataset.

of runtime by reducing read/write I/O time, and outperforms Apache Spark based ADAM Cannoli BWA-MEM by 2x in terms of runtime.

Short-reads variant calling workflow: As discussed in the alignment section, we store BWA-MEM SAM output in chromosomes files. Sorting can be performed using existing tools on these chromosome files in parallel on the cluster. We use *Samtools'* sorting algorithm since it is one of the most efficient due to its in-memory and multi-threading functionality. Similarly, for duplicate removal we use *Picard*'s MarkDuplicate compatible algorithm *Sambamba*, which is also multi-threaded. We also use GATK best practices pipeline applications like *Base quality score recalibration* (BQSR), *ApplyBQSR*, and *HaplotypeCaller* afterwards. Moreover, we integrate *DeepVariant* and *Octopus*, both recent and accurate variant callers, in this workflow. Their published results show high accuracy and F_1-score compared to other state-of-the-art variant callers like *GATK4 HaplotypeCaller* [19], *Strelka2* [42], or *FreeBayes* [43].

We compare the total runtime of these workflows on varying numbers of nodes. As shown in Figure 6.3, moving from a single node to two nodes a more than 3x runtime speedup is achieved because we allocated multiple MPI processes for pre-processing and variant calling applications on each node as these tools have some single node (128 CPU cores) scalability limitations. Increasing the cluster size from 4 to 8 nodes yields only 70-80% runtime improvements because of poor chromosome load-balance. The same runtime speedup is observed for DeepVariant instead of Octopus. On the other hand, GATK best practices pipeline applications are either slow or single-threaded; therefore we only focus on optimizing the workflow by insuring accuracy while running with maximum efficiency on a single node. As shown in Figure 6.4, using alternative application like Samtools/Sambamba instead of Picard (sort and markdup) respectively, and parallel execution of GATK BQSR, ApplyBQSR and HaplotypeCaller applications for chromosomes we have achieved more than 6.5x runtime speedup compared to the baseline. Since GATK processes chromosomes using a single thread, distributing it's execution across multiple nodes is not worthwhile due to the limited number of chromosomes. We have only 25 chromosomes that can be ran in parallel on a single node more efficiently.

Long reads alignment: GenMPI is the first-ever cluster scale implementation of any long reads aligners. As discussed in Section 6.3, both chromosomes based SAM files and a single SAM file output options are implemented in Minimap2. Figure 6.5 shows the total runtime for long-reads aligner, Miniamap2, which exhibits close to linear scalability when increasing the number of nodes in the cluster. The ideal theoretical values for linear scalability are also shown. As shown in the figure, there is a minor performance degradation compared to ideal runtimes that is caused by read/write I/O operations for long reads. Due to the efficient performance of network-attached parallel file system, the overhead of storing SAM data in multiple chromosomes file is minimal compared to a single SAM file generation from all the MPI processes.

Long reads variant calling workflow: Almost all new long reads variant callers only require aligned and sorted reads. Long reads sorting is performed

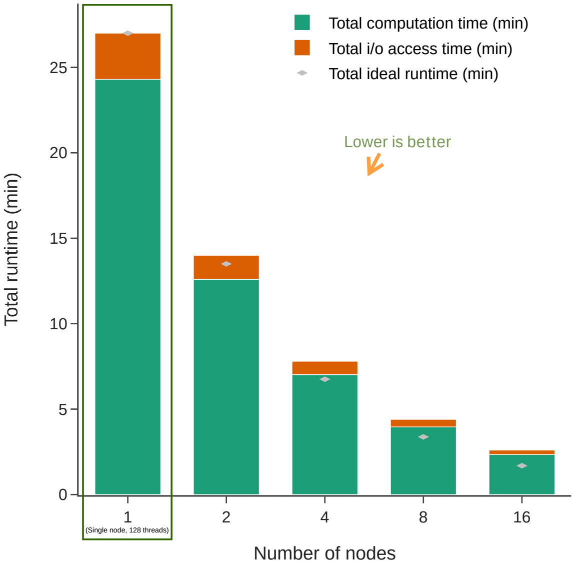


Figure 6.5 Run-time and scalability performance of MPI based Minimap2 implementation. HG002(NA24385) PacBio HiFi 35x coverage dataset has been used on a cluster of varying number of nodes.

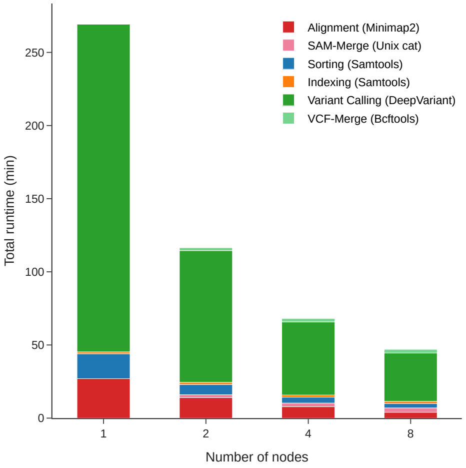


Figure 6.6 Run-time and scalability benchmark of this MPI based implementation for Minimap2 aligner and DeepVariant variant caller on a cluster of varying number of nodes using HG002 (NA24385) PacBio HiFi 35x coverage dataset.

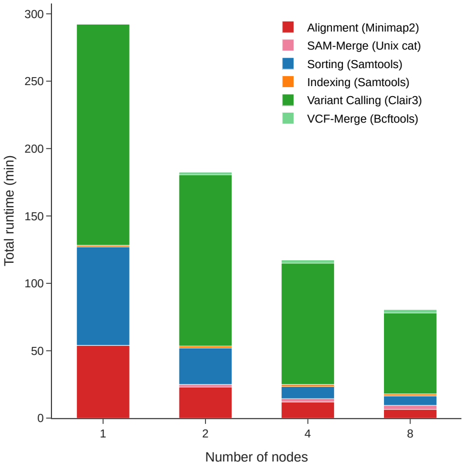


Figure 6.7 Run-time and scalability benchmark of this MPI based implementation for Minimap2 aligner and Clair3 variant caller on a cluster of varying number of nodes using HG002 (NA24385) ONT Guppy 3.6.0 dataset.

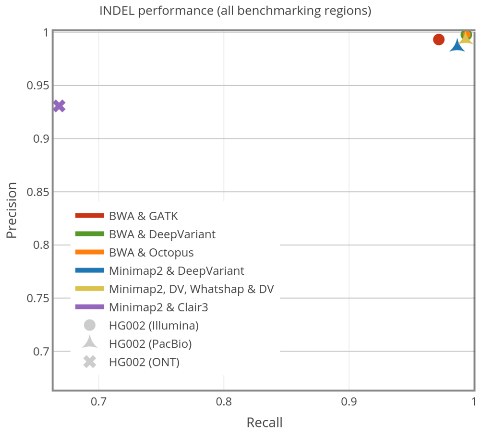


Figure 6.8 INDEL performance comparison of different short and long reads variant calling methods on HG002 dataset.

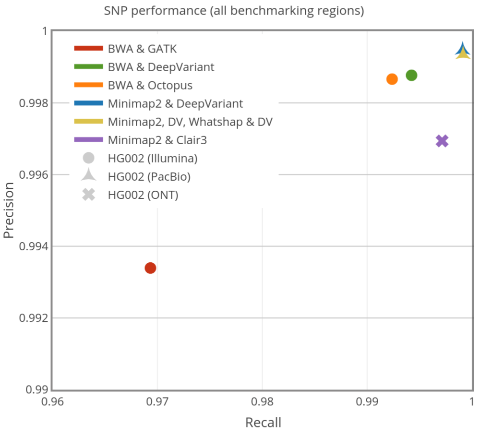


Figure 6.9 SNP performance comparison of different short and long reads variant calling methods on HG002 dataset.

through Samtools 'sort' algorithm. As mentioned before, for PacBio data, we used DeepVariant and DeepVariant with WhatsHap [22]. WhatsHap is used to reconstruct the chromosomes haplotypes and then write out the input VCF augmented with phasing information in its first pass, in the second pass WhatsHap haplotag writes information of reads along with the variants. This information can be used again in DeepVariant for better INDEL detection and accuracy purposes. In Figure 6.6, we show the scalability results of long reads variant calling workflow (Minimap2, Samtools sorting and DeepVariant) on a cluster where more than 2.25x runtime speedup is achieved from 1 node to 2 node cluster and a total of 5x speedup is achieved on a 8 node cluster. This scalability is again only valid up to 8 nodes since we aim at reproducing the exact same variant output as that of a single node, which means that we do not sub-divide chromosomes into smaller pieces. Similarly, for ONT dataset we used Clair3 which provides better accuracy and performance for both SNP and INDEL variants as compared to other variant callers on ONT data. As shown in Figure 6.7 due to some internal scalability limitations of Clair3 we are only able to achieve 1.5x runtime speedup for two nodes cluster as compared to a single node runtime and similar limited scalability trend is shown for more nodes.

6.4.2. Accuracy

In this subsection, we present small variants (SNP and INDEL) detection accuracy. The GA4GH small variant benchmarking tool hap.py has been used to compare the variants in all methods. We compared all these statistics on Chr1-22 and X, Y for each dataset. We compared the small variants (SNP and INDEL) detection accuracy for both short and long reads on HG002 (PrecisionFDA challenge V2) dataset against GIAB v4.2.1 benchmark set. We only tested the accuracy metrics for all-benchmarking region of GIAB v4.2.1 benchmark.

SNP accuracy

Figure 6.8 shows the accuracy performance of SNPs in terms of precision and recall for short reads methods (GATK HaplotypeCaller, Octopus and DeepVariant) on Illumina dataset and long reads methods (DeepVariant, DeepVariant with WhatsHap and Clair3) on PacBio and ONT datasets respectively. Both DeepVariant and DeepVariant with WhatsHap on PacBio data perform best as compared to other methods while GATK HaplotypeCaller SNP performance is well below all other methods. Overall, the best SNP F_1 score is 0.999283 for PacBio dataset using DeepVariant variant caller, as shown in Table 6.2.

INDELs accuracy

Similarly in Figure 6.9 the accuracy performance in terms of precision and recall of INDELs for short reads methods (GATK HaplotypeCaller, Octopus and DeepVariant) on Illumina dataset and long reads methods (DeepVariant, DeepVariant with WhatsHap and Clair3) on PacBio and ONT datasets respectively has been shown. We have observed for INDEL performance that Illumina short read dataset seems to performs best as compared to other datasets for both DeepVariant and Octopus methods. ONT dataset INDEL performance is lower than any other datasets. Overall the best INDEL F_1 score is 0.995957 for Illumina dataset using Octopus variant caller as shown in Table 6.1.

6.5. Discussion

This section further elaborates on the benefits of this approach in a broader context of its applicability in real-time usage on HPC clusters.

6.5.1. Runtimes

Total runtimes of the used aligners show a good linear scalability on small to big clusters in both single SAM output as well as for chromosomes based SAM output. Similarly, for complete variant calling workflows almost all methods provide more than 2x to 6x lower runtime when executed on 2 to 8 cluster nodes, respectively. Because GATK best practices pipeline applications are not multi-threaded,

Table 6.1 Accuracy evaluation of small variants for HG002 (pair-end Illumina short reads) against GIAB HG002 v4.2 benchmarking set for different methods adopted in this workflow. This table shows the SNP and INDEL results for all benchmarking regions.

Method	Variant type	Truth total	TP	FN	FP	Recall	Precision	F_1-Score
DeepVariant	INDEL	525469	522169	944037	1238	0.993720	0.997731	0.995721
-	SNP	3365127	3345702	3815162	4125	0.994228	0.998769	**0.996493**
GATK	INDEL	525469	510602	913193	3671	0.971707	0.993147	0.982310
-	SNP	3365127	3261925	3800121	21703	0.969332	0.993393	0.981215
Octopus	INDEL	525469	522366	935701	1195	0.994095	0.997825	**0.995957**
-	SNP	3365127	3339460	3719754	4455	0.992373	0.998662	0.995507

Table 6.2 Accuracy evaluation of small variants for HG002 (PacBio) against GIAB HG002 v4.2 benchmarking set for different methods adopted in this workflow. This table shows the SNP and INDEL results for all benchmarking regions.

Method	Variant type	Truth total	TP	FN	FP	Recall	Precision	F_1-Score
DeepVariant	INDEL	525469	518336	970119	7560	0.986425	0.986187	0.986306
-	SNP	3365127	3362077	4054200	1778	0.999094	0.999472	**0.999283**
DV-WH-DV	INDEL	525469	521828	981392	3844	0.993071	0.992977	**0.993024**
-	SNP	3365127	3362239	4060995	2199	0.999142	0.999347	0.999244

Table 6.3 Accuracy evaluation of small variants for HG002 (ONT) against GIAB HG002 v4.2 benchmarking set for different methods adopted in this workflow. This table shows the SNP and INDEL results for all benchmarking regions.

Method	Variant type	Truth total	TP	FN	FP	Recall	Precision	F_1-Score
Clair3	INDEL	525469	351200	554604	26874	0.668355	0.930726	**0.778016**
-	SNP	3365127	3355502	4148864	10306	0.997140	0.996939	**0.997039**

cluster scalability is not possible. Neverthelesse, we have achieved more than 6.5x speedup when executing in parallel on a single node. Similarly, Clair3 has multi-threading limitations that preclude proper scaling across multiple nodes of a cluster.

6.5.2. Accuracy and reproducibility

Accuracy results for all benchmarking regions (Chr1-Chr22, X and Y) for the Illumina short-read dataset using GATK HaplotypeCaller, Octopus, and DeepVariant variant callers are shown in Table 6.1. Similarly, Table 6.2 shows accuracy results for the PacBio dataset using the DeepVariant and DeepVariant with WhatsHap variant callers. Table 6.3 lists the accuracy results for the ONT dataset using the Clair3 variant caller. These results are obtained through GA4GH small variant benchmarking tool hap.py.

The accuracy comparisons between the MPI versions of the pipelines and their single node baseline show that these workflows produce the exact same recall, precision, and F_1-score for both short and long reads based variant calling workflows. This ensures the reproducibility of original BWA-MEM and Minimap2 functionality when using GenMPI.

6.5.3. Scalability

The MPI parallelization of both aligners (BWA-MEM and Minimap2) is highly scalable. To further evaluate the scalability of both aligners, we tested the GIAB 300x coverage whole genome sample NA12878 for HG001 with the GenMPI using BWA-MEM. Figure 6.10 shows scalability results of aligning almost 2.5 TBytes data in just 10 minutes walltime on a 64 nodes. We show both computation and I/O time for both aligners separately. In both aligners, the resultant graphs show that increasing the number of nodes in the cluster, the runtime of alignments decreases almost linearly. We also observed that I/O time is increasing slightly

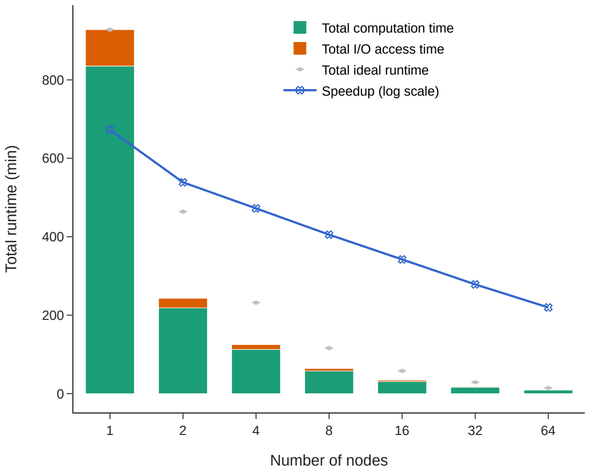


Figure 6.10 GenMPI BWA-MEM performance and scalability measurements for 300x sequencing coverage WGS data from Genome in a Bottle (GIAB) aligned with novoalign for the Illumina HiSeq 300x reads for NA12878 [39].

when increasing the number of nodes but it is not a bottleneck for the scalability.

The only issue is in shared MPI I/O, particularly in writing and reading part of Minimap2 for long reads is a limiting factor for linear scalability. Both MPI I/O and POSIX file system options to output a single SAM file, chromosomes-based SAM output and chromosomes-regions specific SAM output have been implemented as shown in Figure 6.11. We have observed significant overhead due to synchronization effects in writing to POSIX I/O as well as MPI I/O when a single output SAM file is written, as shown in Figure 6.11(a). For per-chromosome SAM output (Figure 6.11(b)), the best suitable option is using POSIX I/O. For the chromosomes-regions specific SAM output option (Figure 6.11(c)), we used 128 output files, which provides comparable results to per-chromosome SAM output with POSIX I/O but still affects the overall performance compared to ideal runtime. The reason behind this is inner reads identification and writing loops which direct reads to a specific file. We have tested both MPI I/O blocking and non-blocking I/O operations for writing SAM output. Due to processes synchronization for wiring SAM data chunks to I/O, an extra overhead slows down the writing of results. This overhead can be mitigated by directly integrating aligners with sorting and indexing applications without relying on file I/O. This will be part of our future work (Section 6.5.7).

6.5.4. Portability and deployment

By using the MPI standard, this workflow is portable and easily deployable to any HPC cluster. A detailed description and quick start guide to run all methods

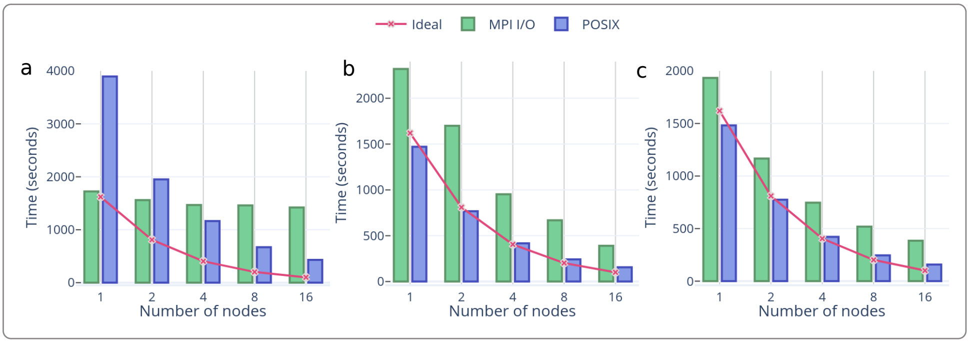


Figure 6.11 Performance and scalability comparisons when using MPI shared I/O and POSIX I/O for (a) single SAM output (b) chromosomes based SAM output and (c) chromosomes regions specific SAM output.

in this approach are given on the project github page. We also tested this implementation with OpenMPI, Intel MPI and HPE MPI flavors and it compiles/runs appropriately.

6.5.5. Cost efficiency

Our cost estimations predict that a significant amount can be saved when opting for public clouds clusters instead of a single large node in the cloud. Particularly in BWA-MEM alignment, clusters utilize maximum system resources resulting in cost saving of more than 50%. Similarly, DeepVariant and Octopus variant callers have some limitation in single node performance for a higher number of cores per socket. More than 20-30% cost can be saved using these variant callers on a cluster with each node having two sockets each with 64 CPU cores.

6.5.6. Memory consumption

MPI based scalable implementations can have a large edge over Apache Spark based variant calling workflows in terms of memory consumption. The MPI implementations do not need extra memory for the platform itself nor to store the data during data shuffling in Spark based implementations.

6.5.7. Future work

This whole workflow implementation (GenMPI) uses storage for intermediate applications I/O read and write operations in form of SAM/BAM files. For future work, integrating pre-processing applications like GenMPI BWA-MEM with sorting (mpiSORT [41]) and index generation would yield further performance gains. We have observed that 5-10% of the total time in GenMPI BWA-MEM and 50% of the total time in mpiSORT are spent on file I/O, which could be mitigated through this approach.

6.6. Conclusion

In this work, we have implemented GenMPI, an MPI based scalable method for both widely used short and long reads aligners, BWA-MEM and Minimap2, respectively. One of the main goals of GenMPI is to ensure 100% identical variant output compared to the single node baseline. In addition, GenMPI provides a flexible architecture which can be used to integrate a variety of alignment and variant calling tools. Compressed and uncompressed FASTQ input is supported and output can be a single SAM file or chromosomes-based SAM files generated by MPI processes. This output can be stored on network attached storage through both POSIX and MPI shared I/O, whichever is convenient and efficient for the underlying HPC system. Results show that these implementations outperform existing Apache Spark based implementation of alignment algorithms by 2x and yield a 20% speedup over state-of-the-art MPI implementations of the BWA-MEM algorithm. Likewise, we also integrate pre-processing applications (sorting, indexing, duplicates removal (for short reads)) as well as variant callers like GATK HaplotypeCaller, DeepVariant, Octopus and Clair3 for both short (Illumina) and long reads (PacBio and ONT) datasets. The variant calling workflows are scalable for up to 8 nodes cluster while giving 2x to 6x total runtime speedups.

Particularly, we show that the distribution of chromosomes across aligners is almost linearly scalable, even when tested using 300x coverage datasets on up to 64 nodes cluster for BWA-MEM. At scale, the alignment completes in under 10 minutes walltime. Thanks to the use of MPI, the workflow implementation is portable and easily deployable on any public cloud or private HPC cluster with minimal effort. Memory requirements do not exceed the actual software needs. The final accuracy results (Recall, Precision, and F_1 score) for variant calling have been reproduced with hap.py against latest GIAB benchmarks set and are shown to be identical to those of the baseline pipeline.

REFERENCES

[1] J. M. Zook and et al. “Extensive sequencing of seven human genomes to characterize benchmark reference materials”. In: *Scientific Data* 3.1 (June 2016), p. 160025. issn: 2052-4463. doi: 10.1038/sdata.2016.25. url: https://doi.org/10.1038/sdata.2016.25.

[2] e. a. Kline. “Targeted next-generation sequencing of pediatric neuro-oncology patients improves diagnosis, identifies pathogenic germline mutations, and directs targeted therapy”. In: *Neuro-Oncology* 19.5 (Nov. 2016), pp. 699–709. issn: 1522-8517. doi: 10.1093/neuonc/now254. eprint: https://academic.oup.com/neuro-oncology/article-pdf/19/5/699/17687132/now254.pdf. url: https://doi.org/10.1093/neuonc/now254.

[3] N. Becher, L. Andreasen, P. Sandager, S. Lou, O. B. Petersen, R. Christensen, and I. Vogel. “Implementation of exome sequencing in fetal diagnostics—Data and experiences from a tertiary center in Denmark”. In: *Acta Obstetricia et Gynecologica Scandinavica* 99.6 (2020), pp. 783–790. doi: https://doi.org/10.1111/aogs.13871. eprint: https://obgyn.onlinelibrary.wiley.com/doi/pdf/10.1111/aogs.13871. url: https://obgyn.onlinelibrary.wiley.com/doi/abs/10.1111/aogs.13871.

[4] T. S. Roman, S. B. Crowley, M. I. Roche, A. K. M. Foreman, J. M. O’Daniel, B. A. Seifert, K. Lee, A. Brandt, C. Gustafson, D. M. DeCristo, N. T. Strande, L. Ramkissoon, L. V. Milko, P. Owen, S. Roy, M. Xiong, R. S. Paquin, R. M. Butterfield, M. A. Lewis, K. J. Souris, D. B. Bailey, C. Rini, J. K. Booker, B. C. Powell, K. E. Weck, C. M. Powell, and J. S. Berg. “Genomic Sequencing for Newborn Screening: Results of the NC NEXUS Project”. In: *The American Journal of Human Genetics* 107.4 (2020), pp. 596–611. issn: 0002-9297. doi: https://doi.org/10.1016/j.ajhg.2020.08.001. url: https://www.sciencedirect.com/science/article/pii/S000292972030269X.

[5] M. Allegretti, A. Fabi, S. Buglioni, A. Martayan, L. Conti, E. Pescarmona, G. Ciliberto, and P. Giacomini. “Tearing down the walls: FDA approves next generation sequencing (NGS) assays for actionable cancer genomic aberrations”. In: *Journal of Experimental & Clinical Cancer Research* 37.1 (Mar. 2018), p. 47. issn: 1756-9966. doi: 10.1186/s13046-018-0702-x. url: https://doi.org/10.1186/s13046-018-0702-x.

[6] S. Deorowicz and S. Grabowski. “Compression of DNA sequence reads in FASTQ format”. In: *Bioinformatics* 27.6 (Jan. 2011), pp. 860–862. issn: 1367-4803. doi: 10.1093/bioinformatics/btr014. eprint: https://academic.oup.com/bioinformatics/article-pdf/27/6/860/16902604/btr014.pdf. url: https://doi.org/10.1093/bioinformatics/btr014.

[7] Apache. *Apache Spark: Lightning-fast unified analytics engine [Accessed: 2nd April 2019]*. 2019. url: https://spark.apache.org/.

[8] Apache. *Apache Hadoop [Accessed: 2nd April 2019]*. 2019. url: https://hadoop.apache.org/.

[9] J. Dean and S. Ghemawat. “MapReduce: Simplified Data Processing on Large Clusters”. In: *Commun. ACM* 51.1 (Jan. 2008), pp. 107–113. issn: 0001-0782. doi: 10.1145/1327452.1327492. url: http://doi.acm.org/10.1145/1327452.1327492.

[10] M. Massie, F. Nothaft, C. Hartl, C. Kozanitis, A. Schumacher, A. D. Joseph, and D. A. Patterson. *ADAM: Genomics Formats and Processing Patterns for Cloud Scale Computing*. Tech. rep. UCB/EECS-2013-207, EECS Department, University of California, Berkeley, 2013.

[11] H. Mushtaq, N. Ahmed, and Z. Al-Ars. “SparkGA2: Production-quality memory-efficient Apache Spark based genome analysis framework”. In: *PLOS ONE* 14 (Dec. 2019), pp. 1–14. doi: 10.1371/journal.pone.0224784. url: https://doi.org/10.1371/journal.pone.0224784.

[12] T. Ahmad, Z. Al Ars, and H. P. Hofstee. “VC@Scale: Scalable and high-performance variant calling on cluster environments”. In: *GigaScience* 10.9 (Sept. 2021). giab057. issn: 2047-217X. doi: 10.1093/gigascience/giab057. eprint: https://academic.oup.com/gigascience/article-pdf/10/9/giab057/40327053/giab057.pdf. url: https://doi.org/10.1093/gigascience/giab057.

[13] D. Decap, J. Reumers, C. Herzeel, P. Costanza, and J. Fostier. “Halvade: scalable sequence analysis with MapReduce”. eng. In: *Bioinformatics (Oxford, England)* 31.15 (Aug. 2015). btv179[PII], pp. 2482–2488. issn: 1367-4811. doi: 10.1093/bioinformatics/btv179. url: https://doi.org/10.1093/bioinformatics/btv179.

[14] L. X, K. Qiu, P. Liang, and P. D. “Speeding up large-scale next generation sequencing data analysis with pBWA”. In: *Journal of biocomputing* 1 (Jan. 2012). doi: 10.4172/jbcg.1000101.

[15] Lustre. *Lustre parallel filesystem*. 2020. url: https://www.lustre.org/.

[16] IBM. “IBM Spectrum Scale”. In: (2020). url: https://www.ibm.com/products/spectrum-scale.

[17] H. Li and R. Durbin. “Fast and accurate short read alignment with Burrows–Wheeler transform”. In: *Bioinformatics* 25.14 (May 2009), pp. 1754–1760. issn: 1367-4803. doi: 10.1093/bioinformatics/btp324. url: https://doi.org/10.1093/bioinformatics/btp324.

[18] H. Li. “Minimap and miniasm: fast mapping and de novo assembly for noisy long sequences”. In: *Bioinformatics* 32.14 (July 2016). 27153593[pmid], pp. 2103–2110. issn: 1367-4811. doi: 10.1093/bioinformatics/btw152. url: https://www.ncbi.nlm.nih.gov/pubmed/27153593.

[19] B. Institute. *Genome Analysis Toolkit [Accessed: 11 April 2019]*. 2010. url: https://software.broadinstitute.org/gatk/.

[20] D. P. Cooke, D. C. Wedge, and G. Lunter. “A unified haplotype-based method for accurate and comprehensive variant calling”. In: *Nature Biotechnology* 39.7 (July 2021), pp. 885–892. issn: 1546-1696. doi: 10.1038/s41587-021-00861-3. url: https://doi.org/10.1038/s41587-021-00861-3.

[21] R. Poplin, P.-C. Chang, D. Alexander, S. Schwartz, T. Colthurst, A. Ku, D. Newburger, J. Dijamco, N. Nguyen, P. T. Afshar, S. S. Gross, L. Dorfman, C. Y. McLean, and M. A. DePristo. “A universal SNP and small-indel variant caller using deep neural networks”. In: *Nature Biotechnology* 36 (Sept. 2018). url: https://doi.org/10.1038/nbt.4235.

[22] M. Martin, M. Patterson, S. Garg, S. O Fischer, N. Pisanti, G. W. Klau, A. Schöenhuth, and T. Marschall. “WhatsHap: fast and accurate read-based phasing”. In: *bioRxiv* (2016). doi: 10.1101/085050. eprint: https://www.biorxiv.org/content/early/2016/11/14/085050.full.pdf. url: https://www.biorxiv.org/content/early/2016/11/14/085050.

[23] Z. Zheng, S. Li, J. Su, A. W.-S. Leung, T.-W. Lam, and R. Luo. “Symphonizing pileup and full-alignment for deep learning-based long-read variant calling”. In: *bioRxiv* (2021). doi: 10.1101/2021.12.29.474431. eprint: https://www.biorxiv.org/content/early/2021/12/30/2021.12.29.474431.full.pdf. url: https://www.biorxiv.org/content/early/2021/12/30/2021.12.29.474431.

[24] P. Danecek, J. K. Bonfield, J. Liddle, J. Marshall, V. Ohan, M. O. Pollard, A. Whitwham, T. Keane, S. A. McCarthy, R. M. Davies, and H. Li. “Twelve years of SAMtools and BCFtools”. In: *GigaScience* 10.2 (Feb. 2021). giab008. issn: 2047-217X. doi: 10.1093/gigascience/giab008. eprint: https://academic.oup.com/gigascience/article-pdf/10/2/giab008/36332246/giab008.pdf. url: https://doi.org/10.1093/gigascience/giab008.

[25] P. Krusche. *Haplotype VCF comparison tools*. 2021. url: https://github.com/Illumina/hap.py.

[26] GIAB. *GIAB v4.2.1 small variant benchmark for HG002-HG004*. 2021. url: https://ftp-trace.ncbi.nlm.nih.gov/giab/ftp/release/AshkenazimTrio/.

[27] H. Li. "The Sequence Alignment/Map format and SAMtools". In: *Bioinformatics* 25 (Jan. 2009), pp. 2078–2079.

[28] A. Tarasov, A. J. Vilella, E. Cuppen, I. J. Nijman, and P. Prins. "Sambamba: fast processing of NGS alignment formats". In: *Bioinformatics* 31.12 (June 2015). 25697820[pmid], pp. 2032–2034. issn: 1367-4811. doi: 10.1093/bioinformatics/btv098. url: https://www.ncbi.nlm.nih.gov/pubmed/25697820.

[29] medaka. *Medaka - Sequence correction provided by ONT Research*. 2021. url: https://github.com/nanoporetech/medaka#usage.

[30] Slurm. *Slurm workload manager*. 2020. url: https://www.schedmd.com/.

[31] OpenPBS. "OpenPBS workload manager". In: (2020). url: https://www.openpbs.org/.

[32] W. Shen, S. Le, Y. Li, and F. Hu. "SeqKit: A Cross-Platform and Ultrafast Toolkit for FASTA/Q File Manipulation". In: *PLOS ONE* 11.10 (Oct. 2016), pp. 1–10. doi: 10.1371/journal.pone.0163962. url: https://doi.org/10.1371/journal.pone.0163962.

[33] T. Hoefler, J. Dinan, R. Thakur, B. Barrett, P. Balaji, W. Gropp, and K. Underwood. "Remote Memory Access Programming in MPI-3". In: *ACM Trans. Parallel Comput.* 2.2 (June 2015). issn: 2329-4949. doi: 10.1145/2780584. url: https://doi.org/10.1145/2780584.

[34] J. Schuchart, A. Bouteiller, and G. Bosilca. "Using MPI-3 RMA for Active Messages". In: (2019), pp. 47–56. doi: 10.1109/ExaMPI49596.2019.00011.

[35] A. M. Wenger, P. Peluso, W. J. Rowell, P.-C. Chang, R. J. Hall, G. T. Concepcion, J. Ebler, A. Fungtammasan, A. Kolesnikov, N. D. Olson, A. Töpfer, M. Alonge, M. Mahmoud, Y. Qian, C.-S. Chin, A. M. Phillippy, M. C. Schatz, G. Myers, M. A. DePristo, J. Ruan, T. Marschall, F. J. Sedlazeck, J. M. Zook, H. Li, S. Koren, A. Carroll, D. R. Rank, and M. W. Hunkapiller. "Accurate circular consensus long-read sequencing improves variant detection and assembly of a human genome". In: *Nature Biotechnology* 37.10 (Oct. 2019), pp. 1155–1162. issn: 1546-1696. doi: 10.1038/s41587-019-0217-9. url: https://doi.org/10.1038/s41587-019-0217-9.

[36] Hawk. "HPE APOLLO (HAWK): Next-Generation HPC System @ HLRS". In: (2021). url: https://www.hlrs.de/systems/hpe-apollo-hawk/.

[37] SurfSara. *Cartesius: the Dutch supercomputer*. 2020. url: https://userinfo.surfsara.nl/systems/cartesius.

[38] FDA. *PrecisionFDA Truth Challenge V2: Calling Variants from Short and Long Reads in Difficult-to-Map Regions*. 2019. url: https://precision.fda.gov/challenges/10.

[39] GIAB. *NHGRI Illumina 300X BAM*. 2020. url: ftp://ftp-trace.ncbi.nlm.nih.gov/giab/ftp/data/NA12878/NIST_NA12878_HG001_HiSeq_300x/NHGRI_Illumina300X_novoalign_bams/.

[40] UCSC. *GRCh38*. 2019. url: ftp://ftp.ncbi.nlm.nih.gov/genomes/all/GCA/000/001/405/GCA_000001405.15_GRCh38/seqs_for_alignment_pipelines.ucsc_ids/GCA_000001405.15_GRCh38_no_alt_analysis_set.fna.gz.

[41] Frédéric, N. Joly, N. Fedy, T. Magalhaes, L. Sirotti, P. Paganiban, F. Martin, M. McManus, and P. Hupé. “QUARTIC: QUick pArallel algoRithms for high-Throughput sequencIng data proCessing”. In: *F1000Research* 9 (Oct. 2020), p. 240. doi: 10.12688/f1000research.22954.3.

[42] S. Kim, K. Scheffler, A. L. Halpern, M. A. Bekritsky, E. Noh, M. Källberg, X. Chen, Y. Kim, D. Beyter, P. Krusche, and C. T. Saunders. “Strelka2: fast and accurate calling of germline and somatic variants”. In: *Nature Methods* 15.8 (2018), pp. 591–594. issn: 1548-7105. doi: 10.1038/s41592-018-0051-x. url: https://doi.org/10.1038/s41592-018-0051-x.

[43] E. Garrison and G. Marth. *Haplotype-based variant detection from short-read sequencing*. 2012. eprint: arXiv:1207.3907,[Accessed:11April2019].

7
Conclusion & future work

7.1. Dissertation summary

This dissertation presents scalable and high performance computational methods for the optimization of genomics applications and algorithms, particularly for whole genome/exome sequencing, data processing and analysis. We optimized BWA-MEM and Minimap2 sequence alignment algorithms as well as different alternatives for pre-processing tools like Samtools, Sambamba and Picard. In addition, we used innovative high performance computational methods to improve the performance and scalability of variant callers like the GATK HaplotypeCaller, DeepVariant, Octopus and Clair3. These methods focus on many performance optimization aspects, especially in-memory, data-parallel and distributed processing of genomics data. We used extensively both single and multi-node public clouds and HPCs computing systems to test and validate the reproducibility and scalability of these methods.

Human understanding of complex DNA structure and its genes is still in an initial stage. Higher throughput, enhanced accuracy and dropping costs of next and third generation sequencing technologies makes sequence analysis more viable to be adopted in research and development and for clinical and diagnostic applications.

As discussed in Chapter 3, refining and cleaning sequencing data for genome analysis is still a time consuming step. To make efficient use of computing resources in genomics pipelines to process genomics data in a reasonable amount of time is achievable through (i) in-memory fast data access, (ii) optimized inter-process communication by avoiding unnecessary I/Os communication overhead (iii) by employing load-balancing in data-parallel processing and (iv) vectorized execution where-ever possible. Tab-delimited text-based SAM and random access in BAM data formats enable the characteristics of processing independent and parallel tasks on fine-grained data elements or reads without the need of process synchronization or recursively inner-loops rewriting in complex algorithms. Similarly, for HPC clusters and clouds scalability, converting of these data formats to distributed file systems like HDFS and NFS provides fast data access and efficient communication by minimizing the overhead of I/O operations. Apache Arrow is a unified in-memory columnar data format that enables exploiting the performance of modern processor hardware components. Representing and accessing SAM/BAM data through the columnar in-memory data formats is found to be highly efficient for the GATK best practices variant calling pipeline by avoiding intermediate I/O operations, enabling parallel processing on single node and better cache locality exploitation. Previously, Apache Spark based in-memory and distributed data analytics frameworks for genomics applications and pipelines have been explored extensively. However, row-based processing of these frameworks limits both scalability and efficient resource utilization for many algorithms. Leveraging the columnar in-memory format like Apache Arrow in Apache Spark analytics operations show better scalability and highly efficient system resource utilization, particularly vectorized operations and cache locality, this has been discussed in Chapter 4 of this dissertation. Further, extending the Apache Arrow data format for the scalability of clusters for variant calling pipelines, in-conjunction with Arrow Flight, a wire-speed data transfer protocol was used to shuffle and schedule data across cluster nodes. This combination of both the Apache Arrow format and Arrow Flight as data communication service provides high performance and better scalability as compared to Apache Spark based frameworks and comparable performance with respect to MPI implementation as explored in Chapter 5. Higher memory consumption, communication overheads, additional software dependencies/complexities and data format conversion issues in distributed big data frameworks are considered the main bottlenecks for their adaptivity. Parallel programming models like the Message Passing Interface (MPI) provide bare-metal performance for the applications that do not require process level data sharing and synchronization. Both higher scalability and efficient system resource utilization can be improved at the cost of tuning existing algorithms or writing from scratch as described in Chapter 6.

7.2. Research contributions and future work

The propositions presented in Chapter 1 have been discussed extensively in this dissertation. Below, here again we conclude these propositions briefly with a short description for their respective future work.

Columnar in-memory data formats have already proven helpful in database (transactional) and data-analytic systems.

Proposition: *Columnar in-memory data formats can also be beneficial in genome variant calling applications for better cache locality, vectorized operations and parallel execution on modern processors.*

Two major performance bottlenecks in GATK best practices variant calling pipelines have been observed. One is related to I/O overhead (reading/writing and parsing SAM/BAM data) which we incur when running intermediate applications. The second is related to the lack of multi-threading and parallelism in some Picard and GATK applications. To overcome these limitations, we have described how Apache Arrow, a unified in-memory columnar data format and Plasma Object Store (an API inside the Arrow project) can be integrated in all variant calling applications to achieve both almost zero I/O overhead and better parallel performance. Moreover, this integration guarantees better cache locality and efficient vectorized operations utilization in many applications.

- For ***future*** work, variant calling applications could be redesigned to get the maximum benefits of columnar data, cache locality and vectorized operations. Apache Parquet, a compressed columnar storage format, provides distributed and low latency data access and better compression ratios as compared to the BAM format. Parquet could be used as an alternative to Apache Arrow for big genomics datasets on small memory systems.

Columnar in-memory data formats such as Apache Arrow provide an efficient alternative to store and access in-memory data among multiple big data frameworks or applications by avoiding extra (de)-serialization overhead.

Proposition: *These formats should be integrated into big data frameworks like Apache Spark to avoid (de)-serialization overhead and in-memory data access during data transformation and processing when running different applications inside these frameworks.*

We benefited from combining the ease-of-use of the Python programming language, the ease-of-scalability of big data frameworks like Apache Spark and efficient columnar data formats like Apache Arrow to make a user friendly version of the genomics pipelines eco-system. Vectorized and scalable UDFs execution allows creating an even higher performance and efficient version of the pipelines. We analyzed the potential impact of this approach and implemented a whole variant calling pipeline using these technologies. The resultant pipeline is fast, scalable and efficient in its resource utilization.

- For ***future*** work, we could use this combination of technologies to improve the performance of many widely-used variant calling pipelines. In addition, we also observed that the common practice of repeatedly writing intermediate results to BAM files on disk after sorting and duplicate removal applications is computationally expensive. Instead of creating BAMs, we could use dataframes directly in Python-based variant callers like DeepVariant to improve the overall speedup of the complete pipelines.

Arrow Flight provides a high performance and parallel interface for bulk data (Apache Arrow in-memory columnar format) transfers across networks. We benchmark for maximum client-server and cluster throughput for this data communications protocol.

Proposition: *Arrow Flight protocol can achieve high performance data movement throughput for genomics variant calling workflows across computing clusters as compared to big data frameworks such as Apache Spark.*

Arrow Flight allows for effective scalability of genomics variant calling pipelines on large clusters, by eliminating communication overhead which pose a scalability bottleneck. Today, almost all existing frameworks for processing genomics data are built around big data frameworks like Apache Hadoop and Apache Spark.

We have demonstrated the feasibility of a simple and easy-to-use implementation using a cluster workload manager (e.g., SLURM or OpenPBS) as an application handler and data scheduler to replace the Apache Spark framework or MPI based implementation of genomics variant calling pipelines.

- Arrow Flight could reshape ***future*** distributed big data systems, online analytical processing (OLAP), distributed query as well as data warehouse systems. However, Arrow Flight uses a heavy stack of gRPC on TCP protocol. Future work can use more efficient protocols like RDMA to reduce the communication overhead even further. Also Arrow Flight based distributed Apache Arrow data scheduling, compute and query frameworks like DataFusion and Ballista can be used instead of Apache Spark as distributed engines for easy scalability and scheduling of genomics pipelines.

Genomics variant calling applications like genome alignment and variant calling can be performed on data chunks independently without any data communication or synchronization.

Proposition: *MPI can be used to achieve bare-metal performance in sequence alignment algorithms and variant calling workflows overall.*

We have described the methods and algorithmic changes required to integrate MPI in widely used short-reads (BWA-MEM) and long-reads (Minimap2) aligners to gain high performance and efficient cluster scalability. For sorting and duplicate removal, we used existing Samtools and Sambamaba for chromosomal level scalability on clusters. Similarly, variant calling applications like Octopus and DeepVariant have been integrated into an MPI-RMA based queue to process data on clusters.

- A valuable ***future*** contribution would be to keep intermediate data generated by pre-processing applications in in-memory buffers such that variant calling applications can access this data using MPI-RMA interfaces. This avoids intermediate I/O read/write operations for individual applications. Other future work could focus on integrating Apache Arrow Flight into an MPI communication protocol to benefit from the best of both worlds: a high-abstraction level programming interface, combined with a low-level low-overhead communication protocol.

List of Publications

6. T. Ahmad, J. Schuchart, Z. Al-Ars, C. Niethammer, J. Gracia, and H. P. Hofstee. "GenMPI: Cluster Scalable Variant Calling for Short/Long Reads Sequencing Data". In: *[Under-review]* (2022)

5. T. Ahmad, Z. Al-Ars, C. Ma, and H. P. Hofstee. "Communication-Efficient Cluster Scalable Genomics Data Processing Using Apache Arrow Flight". In: *[Under-review]* (2022)

4. T. Ahmad, Z. Al-Ars, and H. P. Hofstee. "Benchmarking Apache Arrow Flight - A wire-speed protocol for data transfer, querying and microservices". In: *ACM SIGPLAN, Benchmarking in the Data Center: Expanding to the Cloud, (BID'22), April 2, 2022, Seoul, Republic of Korea* (2022). doi: 10.1145/3527199.3527264

3. T. Ahmad, Z. Al Ars, and H. P. Hofstee. "VC@Scale: Scalable and high-performance variant calling on cluster environments". In: *GigaScience* 10.9 (Sept. 2021). giab057. issn: 2047-217X. doi: 10.1093/gigascience/giab057. eprint: https://academic.oup.com/gigascience/article-pdf/10/9/giab057/40327053/giab057.pdf. url: https://doi.org/10.1093/gigascience/giab057

2. T. Ahmad, N. Ahmed, Z. Al-Ars, and H. P. Hofstee. "Optimizing performance of GATK workflows using Apache Arrow In-Memory data framework". In: *BMC Genomics* 21.10 (Nov. 2020), p. 683. issn: 1471-2164. doi: 10.1186/s12864-020-07013-y. url: https://doi.org/10.1186/s12864-020-07013-y

1. T. Ahmad, N. Ahmed, J. Peltenburg, and Z. Al-Ars. "ArrowSAM: In-Memory Genomics Data Processing Using Apache Arrow". In: *2020 3rd International Conference on Computer Applications Information Security (ICCAIS)*. 2020, pp. 1–6. doi: 10.1109/ICCAIS48893.2020.9096725

Acknowledgements

First and foremost, I am grateful to Zaid Al-Ars for giving me the opportunity to conduct PhD research under his supervision. Zaid, your dedication and compassion in my mentorship enabled me to excel in generating valuable research ideas, materializing those ideas, making collaborations, writing and presenting articles. It was always fun to discuss new ideas and technological advancements with you. You provided me the freedom in selecting the path I wanted to explore but after tough debates and discussions. I also learned valuable real-life lessons from you beyond this research. Thank you again for being so kind and generous to me.

I really feel lucky having Peter Hofstee as my teacher and mentor during this PhD journey. Peter, your critical analysis of my papers not only helped in improving them but also provoked me to think differently and smartly. I am indebted to you for your time particularly rephrasing and shaping many sections of this dissertation. Finally, I would say, in the future, when I pursue a career in research, I would like to take you as role model.

I would like to thank my committee members Marcel Reinders, Koen Langendoen, Keijo Tapio Heljanko, Frank Liu and José Gracia for taking the time to review my dissertation and being part of my defense committee.

My gratitude to the Graduate School officials and instructors for their enormous support and guidance in this whole graduation process.

My sincere thanks to the Punjab Educational Endowment Fund for generously funding my PhD program.

I feel fortunate for having Imran, Hamid and Nauman, as my office mates at the Q&CE dept. during the first few months of my PhD. I really enjoyed their discussions and appreciate their help in eventuating my research direction. Nauman, my first day in Delft when you generously accompanied me and showed me around Delft centrum, is an unforgettable memory for me. Your solution-oriented approach and deep understanding of genomics helped me to understand many complicated concepts and algorithms quickly. The credit of timely completion of my PhD goes to you for all the support and motivation you provided me throughout this journey. You were always there during tough times as a source of inspiration and courage.

I would like to say a special thanks to Hamid, for all the help you provided me to understand Apache Spark and your genomics framework, which eventually helped me to create more scalable frameworks. Johan, I sincerely want to thank you for introducing me to the Apache Arrow framework. I really appreciate your gracious support from writing the first few paragraphs of our first paper to those brain-storming sessions of multitude of ideas. BBQs parties at your home and the hospitality from Emma are few of the unforgettable memories. Joost, it was always a pleasure to interact with you and to talk to you occasionally about computing related topic to acquire deep insights. Baozhou, we learned and grew together at the Q&CE department. Spending time with you was always fun. We are seperated apart by the Himalayas but we are close at heart :-).

I am also thankful to the Q&CE dept. faculty specially Koen Berterls and colleagues Jeroen, Motta, Jian, Aritra, Rajendra, Mahdi, Taha, and Abdullah. Jakoba, I thought you were the missing element in our group, and your arrival filled that gap by initiating group activities, which served as a bridge to hold everyone together through those difficult lock-down times. Muath, I missed your presence a lot when you left and decided to pursue some new endeavors. Abdulqader, thank you for always being there for healthy discussions.

I want to thank Chengxin, Cao and Tianli, my master thesis students for working with me on some emerging topics and producing valuable results that helped progress this thesis. I also want to send a special word of appreciation to out system administrator Erik, without you most of the work in our department would not be possible. I also thank the administrative support provided by Lidwina, Joyce, Trisha and Paul at the Q&CE department.

I also want to express my gratitude to Jose and Joseph (HLRS, Germany) for providing me the opportunity to work with them. Joseph, your continuous support and dedication helped me to learn many perspectives of MPI programming in just a few months. Frederic, (Institut Curie, France) thank you for having an open-heart and engaging in discussions that helped me to improve my skills. We will continue our collaboration in the future on the various topics we are working on.

I would like to say a big thank you to my overseas friends; Asif (TU Dresden), Waqas (GCU Glasgow) and Tanweer (Aalto University) for always being there for thoughtful and research related discussions and stress reliving, hours-long talks.

I would also like to thank all my Pakistani friends in Delft with whom I spent joyful times, particularly Aftab, Mohsin, Irfan, Saad, Aitzaz, Qasim, Hussam, Nauman, Noor, Raihan, Samad, Shozab, Yasir, Ahmad, Osama, Haider, Usman, Mubariz, Hassan and Ammar. I want to write a thank you to each one of you separately, but afraid this will go beyond the couple of pages I have. Aftab and Mohsin, special thanks for always being there in terms of emotional and logistic support. Haider, Usman, Mubariz and Hassan, though we met for a brief time in Delft but your company and presence was always a fresh breeze for me. Osama, I always enjoyed your gracious hospitality and your thought provoking discussions which were a treat for mind and heart. Thank you!

Mahmoud and Khalid thanks for spending many beautiful moments together once in a while.

Here I also want to remember and extend my highest gratitude to my undergraduate teacher and mentor Ehsan Khalil, who enlightened the beacon of knowledge in my heart.

I cannot express in words my indebtedness to my parents. My father's and mother's immense love for knowledge and their continuous struggle and sacrifices for us to achieve the best education, your dedication is exemplary. Ammi, Abbu, I could not have completed this PhD without you. Your love and prayers are always the most invaluable support for me and I need them to support me throughout my life.

Last but not least, thanks to my family. My sister Asma particularly, you are the source of motivation and courage not just for me but for everyone around you. Special thanks to my siblings Rozina, Shakeel, Nadeem, Navid, Mehru, Tahira and Touqeer for being so friendly, joyful and supportive all the time, which makes facing real life challenges a lot easier.

Fouzia, I would like to express my sincere gratitude for taking care of me, our home and Husayn Faateh during all these times. You being an encouraging partner and having patience inspired me to look for far-beyond goals. Faateh is a blessing from Allah Almighty who we received during these times. I pray for him to become a source of light, wisdom, knowledge and peace for humanity.

Curriculum Vitæ

Tanveer AHMAD

19-03-1985	Born in Karor Lal-Eason (Layyah), Pakistan.

Education

2004–2009	B.Sc. Computer Engineering Bahauddin Zakariya University Multan, Pakistan
2012–2016	M.Sc. Computer Engineering University of Engineering and Technology Lahore, Pakistan
2018–2022	PhD Computer Engineering Delft University Technology Delft, The Netherlands

Work

2010–2011	Visiting Lecturer Bahauddin Zakariya University, Multan, Pakistan
2011–2017	Embedded Systems Engineer Powersoft19, Lahore, Pakistan
2017–2017	Guest Researcher, CFAED Dresden University of Technology, Germany
2021–2021	Intern, Scalable Programming Models & Tools group High Performance Computing Center Stuttgart (HLRS), Germany