ArrowSAM: In-Memory Genomics Data Processing through Apache Arrow Framework

Tanveer Ahmad
Dept. of Quantum and Computer Engineering
Delft University of Technology
Delft, Netherlands
t.ahmad@tudelft.nl

Nauman Ahmed
Dept. of Quantum and Computer Engineering
Delft University of Technology
Delft, Netherlands
n.ahmed@tudelft.nl

Johan Peltenburg
Dept. of Quantum and Computer Engineering
Delft University of Technology
Delft, Netherlands
j.w.peltenburg@tudelft.nl

Zaid Al-Ars
Dept. of Quantum and Computer Engineering
Delft University of Technology
Delft, Netherlands
z.al-ars@tudelft.nl

***Abstract*—The rapidly growing human genomics data driven by advances in sequencing technologies demands fast and cost-effective processing. However, processing this data brings some challenges particularly in selecting appropriate algorithms and computing platforms. Computing systems need data closer to the processor for fast processing. Previously, due to the 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 commonly used genomics data format, Sequence Alignment/Map (SAM) can be presented in the Apache Arrow in-memory data representation to take benefits of in-memory processing to ensure the better scalability through shared memory Plasma Object Store by avoiding huge (de)-serialization overheads in cross-language interoperability. To demonstrate the benefits of such a system, we presented an in-memory SAM representation, we called it ArrowSAM, Apache Arrow framework is integrated into genome pre-processing applications including BWA-MEM, Sorting and Picard as use cases to show the advantages of ArrowSAM. Our implementation comprises three components, First, We integrated Apache Arrow into BWA-MEM to write output SAM data in ArrowSAM. Secondly, we sorted all the ArrowSAM data by their coordinates in parallel through pandas dataframes. Finally, Apache Arrow is integrated into HTSJDK library (used in Picard for disk I/O handling), where all ArrowSAM data is processed in parallel for duplicates removal. This implementation gives promising performance improvements for genome data pre-processing in term of both, speedup and system resource utilization. Due to columnar data format, better cache locality is exploited in both applications and shared memory objects enable parallel processing.

***Index Terms*—Genomics, Whole Genome/Exome Sequencing, Big Data, Apache Arrow, In-Memory, Parallel Processing**

I. Introduction

Genomics is projected to become the field that generates largest big data sets globally, which requires modifying existing tools to take advantage of new developments in memory technologies to ensure better performance and high throughput. At the end of first ever Human Genome Project [1990—2003], a final draft sequence of the euchromatic portion of the human genome containing approximately 2.85 billion nucleotides was announced [1]. This project also identified 20,000—25,000 human protein-coding genes. Since then, genomics data has been increasing rapidly due to 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 kilo-base-pairs (kbp) in length, such as Oxford Nanopore technologies (ONT) and Pacific BioSciences (PacBio) Single Molecule Real-Time (SMRT) sequencing technology. The ongoing pace of these technologies promises even more longer reads of $\sim$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 [2]. Unfortunately, currently available long read sequencing technologies produce data with high error rates. ONT generates data with mean error rate of $\sim$40% and $\sim$15% for PacBio as compared to Illumina short reads with less than 2% error rate [3]. With the continued decrease in price of DNA sequencing, the bottleneck shifts from generating DNA data to the computation and analysis of DNA information, which needs to keep pace of the high throughput of sequencing machines, and at the same time account for the imperfections of generated data. New applications using DNA data are becoming ever more complex such as the study, 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 [2]. This underscores the need for efficient and cost effective DNA analysis infrastructures.

New storage-class memory (SCM) technologies will soon replace the existing long latency and block-based data transfer hard disk drives and solid-state drive storage mediums. 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 [4]. These memories have a higher latency as compared to DRAM but provide huge capacity and byte addressability at lower costs. Though this memory technology is not an immediate replacement of main memory, but the new features it provides, make it usable in-conjunction with storage and memory, as an additional intermediate tier of memory hierarchy.

A. Problem Definition

Bioinformatics 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 systemswhich incur a huge I/O bottleneck in computation due to disk bandwidth [5]. 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 cannot exploit modern hardware features like multi-core parallelization, Single instruction, multiple data (SIMD) vectorization and accelerators (GPU or FPGAs) performance very well.

B. Contributions

The main contributions of this work are as follows:

- In-memory SAM data representation (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.
- Existing widely used genomics data pre-processing (for alignment, sorting and duplicates removal) applications are integrated into Apache Arrow framework to exploit the benefits of immutable shared memory plasma objects in inter process communication.
- A fair comparison of different workflows for genome pre-processing, which use different techniques for in-memory data communication and placement (for intermediate applications) is presented to show how in-memory columnar data representation outperforms memory based file systems like ramDisk and pipes.

II. Background

This section provides a short description of DNA sequence data pre-processing, variant calling analysis and tools used for this purpose. A brief introduction to the Apache Arrow framework, we use for in-memory SAM data representation and its Plasma shared memory API is also given.

1) Pre-processing/Cleaning: Alignment tools align reads to the different chromosomes of a reference genome and generate an output file in the SAM format, describing various aspects of the alignment result, such as map position and map quality. SAM format 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. 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. Samtools [6], Picard [7], Sambamba [8] and Samblaster [9] are some tools commonly used for such operations.

2) Variant Discovery: Variant discovery is the process of identifying the presence of single nucleotide variants (SNVs) and insertions and deletions (indels) in individual genome sequenced data, these variants may be germline variations (the variations in an individual's DNA inherited from parents) or somatic mutations (the variations occur in cells other than germ cells, which can cause caner or other diseases). GATK and Freebayes are commonly used open-source tools for such analysis. The output of these tools is generated in the variant calling format (VCF) to visualize and further analyze the detected variations.

3) Variant Calling/Analysis Tools: There are many commercially and open-source tools available to build whole genome/exome analysis pipeline. Some of these tools include: Galaxy [10], [11], SevenBridges [12], GenomeNext [13], SpeedSeq [14], BcBioNextgen [15], DNAp [16], and GATK [17] best practices pipeline. GATK is one of the most popular and commonly used pipeline for DNA variant calling.

A. Apache Arrow

To manage and process large data sets, many different 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 computational performance of big data analytics applications. They are generally built on top of high-level language frameworks such as Java and Python. 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.

TABLE I
ARROW BUFFERS LAYOUT

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	1	2019		o	
4	0	null		w	
5				D	
6				a	
7				t	
8				a	
9				!	

TABLE II
RECORD BATCH FOR DATA IN TABLE I

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

TABLE III
SCHEMA FOR RECORD BATCH IN TABLE II

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

While combining different techniques has many advantages, there are also drawbacks. The high amount of heterogeneity causes a requirement of data (de)serialization, whenever the data is moved between components that are implemented in different technologies. An example, where (de)serialization takes place in such a system stack is illustrated in Fig 1.

To mitigate this problem, the Apache Arrow [18] project was initiated by the Apache Foundation in 2016. The main focus of this project is to provide an open standardized format and interfaces for 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 Fig 2. At the time of writing, Apache Arrow supports the following languages: C, C++, C#, Go, Java, JavaScript, MATLAB, Python, R, Ruby, and Rust. Interfaces exist for GPGPU programming, through arrow-cuda-glib. External tools to support FPGA accelerators also exist through the Fletcher project [19].

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. Examples include a Validity bitmap buffer (a single bit stores information related to data value validity i.e, null or valid data in corresponding value buffer), an Offset buffer (offset values for multiple character strings) and Value buffers (stores fixed-width types like in c-like-array). Each RecordBatch contains metadata, called a schema, that represents the data types and names of stored fields in the RecordBatch. An example of RecordBatch with three fields is shown in Table II, their corresponding schema shown in Table III and Arrow Buffers layout in Table I.

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. Moreover, Arrow can efficiently manage the big chunks of memory on its own without any interaction of a specific software language run-time, particularly garbage-collected ones. In this way, large data sets can be stored outside heaps of virtual machines or interpreters, that are often optimized to work with few short-lived objects rather than many objects that are used throughout a whole big data processing pipeline. 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).

B. Plasma In-Memory Object Store

Plasma is an inter-process communication (IPC) component of Arrow, that handles shared memory pools across different heterogeneous systems [20]. 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.

III. MOTIVATION

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

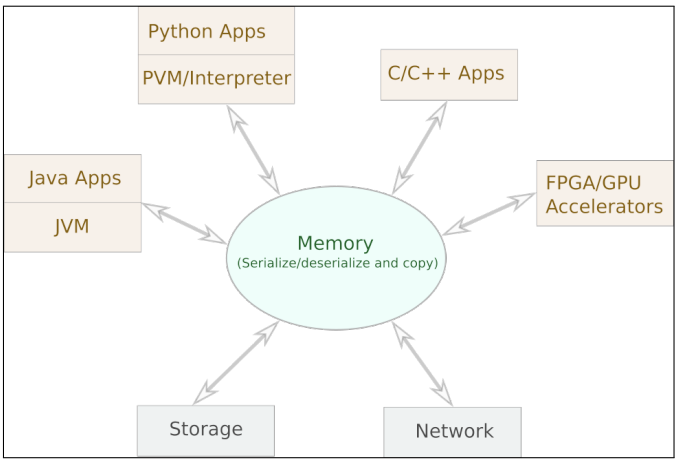

Fig. 1. An example where (de)serialization takes place when data is exchanged between different frameworks and accelerators.

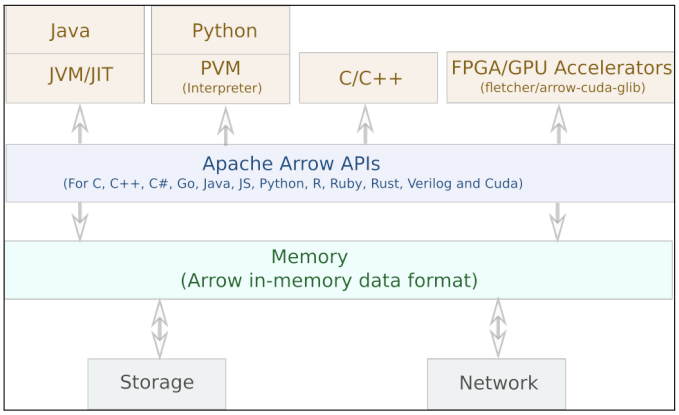

Fig. 2. Apache Arrow provides a unified in-memory format for data placement to be used in many languages and platforms avoiding the (de)serialization overhead.

related to technology. 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. It also keeps data in contiguous and columnar format to exploit cache spatial locality and SIMD vectorization capabilities. For variable length data, Arrow uses offsets instead of lengths for data access, which in turn enables parallel access to such data anywhere in the memory pool. Plasma Object Store provide a lightweight (de)serialization interface for in-memory stored objects. These features make the Arrow framework a good choice for genome data in-memory processing.

From a memory technology point of view, the use of traditional DRAM memory was limited due to cost, volatility and other physical constraints. Therefore, it was not feasible to store large amounts of active data of an application. However, with the emergence of new Persistent and NVM technologies, it becomes feasible to place large sets of active data in memory, close to the processor.

IV. Implementation

In order to enable genomics applications to use Apache Arrow Framework, two different contributions are needed. First, we need to define an Arrow in-memory representation of the corresponding genomics data format. Second, the applications and tools using the data need to be adapted to access the new format as shown in Figure 3 (a). In the following, these two contributions are discussed.

A. In-Memory SAM Format (ArrowSAM)

The SAM file format is an ASCII based, tab delimited text format to represent sequence data shown in Figure 3 (c). It's 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 (d). The data types for each field is stored in schema as shown in the Figure 3 (e). Arrow's tabular data form, called RecordBatches are used to store data in contiguous memory chunks as shown in Figure 3 (b). Each RecordBatch is a combination of a schema, which specifies the types of data fields of the SAM record and the data itself.

B. BWA-MEM Integration

`BWA-MEM` aligns the raw FASTQ read sequences against a large reference genome such as that of a human. We used the in-memory format to store the mapping data produced by `BWA-MEM` from query and reference genome files (FASTQ and FASTA). 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.

C. 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`, SAMTools and `Sambamba` are used to sort the SAM file according to the 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 of 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.

D. 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)`

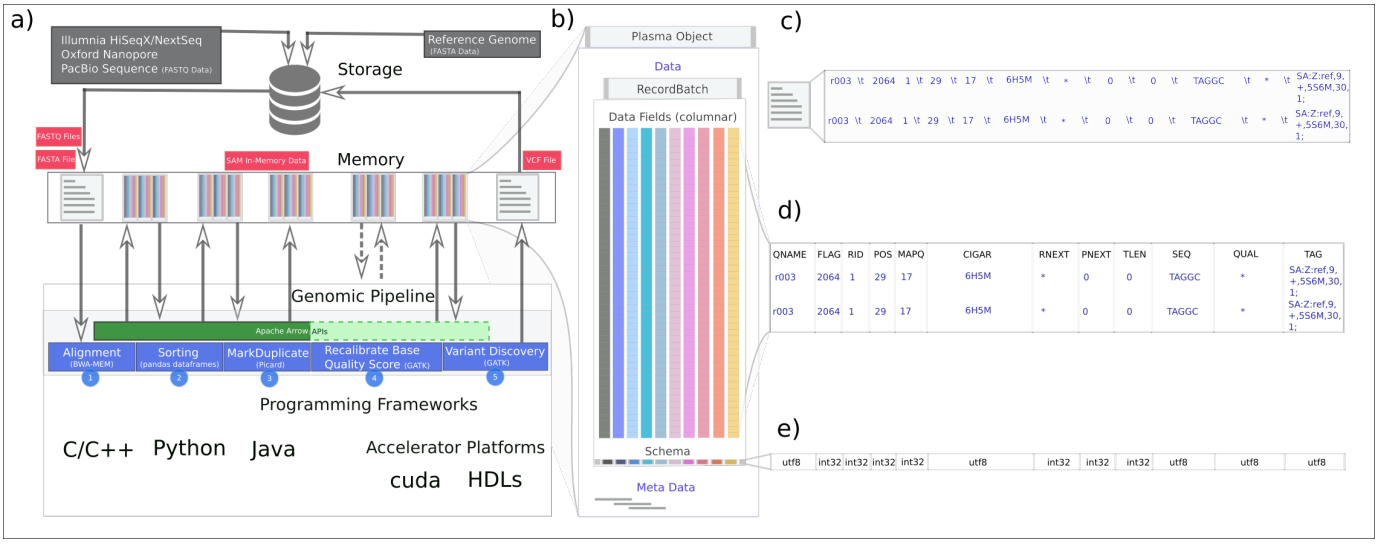

Fig. 3. a) Genomics pipeline using ArrowSAM representation for all intermediate steps, this implementation just reads from I/O at the start and writes to I/O at the end. b) Arrow RecordBatch enclosed in Plasma Object Store. c) SAM file in ASCII text format. d) SAM file in RecordBatch format. e) Schema specifies the data types of ArrowSAM.

which are actually needed to perform the MarkDuplicate operation. We 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) and `MarkDuplicate` tool inside the `Picard` to read the data from all the 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.

TABLE IV
TOOLS AND LIBRARIES USED IN THE EXPERIMENTAL SETUP

Tools/APIs	Version	Author
BWA-MEM	0.7.17	Heng Li
Picard	2.18.14	Broad Institute
Sambamba	v0.6.8	Dlang for Bioinformatics
elPrep	v4.1.5	ExaScience
Arrow C/C++/Java	0.11.0	Appache
PyArrow	0.11.0	Appache
Plasma Object Store	0.11.0	Appache

V. EVALUATION

This section evaluates the scalability, throughput and speedup we have achieved for pre-processing of sequence data in sorting and marking duplicates stages against the existing frameworks.

A. Experimental Setup

All the 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 QPI (QuickPath Interconnect) and share memory through NUMA (non-uniform memory access) architecture. A total of 192-GBytes of DDR4 DRAM with 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. The tools used in the whole setup and frameworks/libraries are listed in Table IV with their versions for future reference. We used Perf [21] tools to collect all cache levels and system utilization statistics.

Data Set

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. 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.

Code and Scripts Availability

The code for 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/arrow-gen. The scripts for running all workflows are also available in the same root directory.

B. Performance Evaluation

In this section, we compare our approach with state-of-the-art tools and approaches used for pre-processing of high throughput genomics sequencing data. All speedups are compared for best case scenario, ie. when ramDisk is used

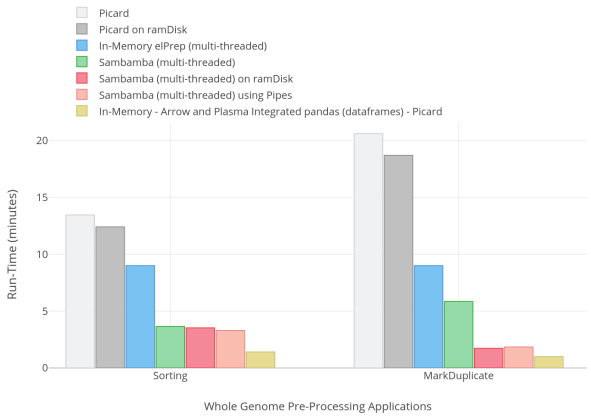

Fig. 4. Execution time of Picard, Sambamba, elPrep and Arrow based Sorting and MarkDuplicate for Whole Genome data set.

for Picard and Smabamba and maximum in-memory settings enabled for elPrep.

1) Picard: A number of `Picard` tools are considered as benchmarks in genome analysis pipelines, such as `Picard MarkDuplicate`. This tool was adapted to use our Arrow in-memory representation. 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/BAM/CRAM files. We modified this tool from two perspectives:

- Instead of reading from and writing to files, it now reads/writes from in-memory RecordBatches, with just those fields/columns which are necessary for `MarkDuplicate` operations.
- `Picard` is single threaded, we changed it to 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` operation instead of parsing all the reads in SAM file. Our implementation gives 8x and 21x speed-ups on `Picard SamSort` for genome and exome data sets respectively. Similarly we achieve 21x and 18x speed-ups on for genome and exome data sets respectively for `Picard MarkDuplicate` stage as shown in Fig 5.

2) Sambamba: is a multi-threaded tool to manipulate SAM/BAM/CRAM files for pre-processing steps in variant calling pipelines. This tool gives linear performance 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 [8]. Our implementation gives 2x and 2x speed-ups on `Sambamba Sort` for genome and exome data sets respectively. Similarly we achieve 1.8x and 3x speed-ups on for genome and exome data sets respectively for `Sambamba MarkDup` stage as shown in Fig 5.

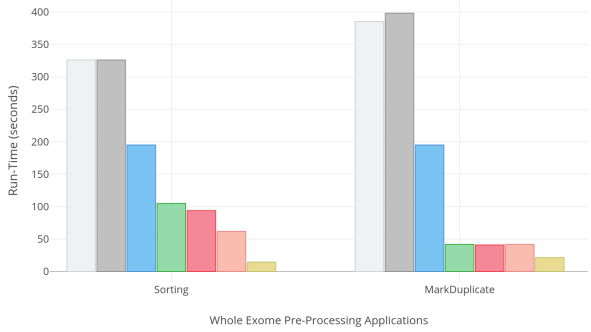

Fig. 5. Execution time of Picard, Sambamba, elPrep and Arrow based Sorting and MarkDuplicate for Whole Exome data set.

3) elPrep: is also a set of multi-threaded tools for pre-processing SAM/BAM files for variant calling in-memory. Despite exploiting the parallelism and in-memory processing, the results reported in paper show a maximum of 13x speedup over GATK best practices pipeline for whole-exome and 7.4x faster for whole-genome while using maximum memory and storage footprints. We have also tested and compared this software with our implementation for pre-processing applications and results show our implementation gives more than 5x speedup over `elPrep`. `elPrep` performs sorting and mark duplicate in a single command, the total run-time for both stages is equally divided in run-time graphs Fig 5.

Samblaster is also used for pre-processing SAM/BAM files which is more fast than `Sambamba`, but has not been considered for performance comparison here because it produces different output for `MarkDuplicate` stage than `Picard` algorithm.

C. Discussion

1) Parallelization and Scalability: In both sorting and duplicates removal stages, the parallelization offered by shared memory plasma objects resulted in a large speedup in overall run time. All 25 chromosomes are sorted and duplicates removed in parallel. The speedup can be improved further by dividing chromosomes further without using any big data frameworks. Figure 7 and Figure 8 shows the CPU utilization in the original `Picard` and in our implementation, respectively for WES. In `Picard` sorting the CPU utilization is poor and the tool is mostly waiting for the 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.

2) Memory Accesses: `Picard` tools read a whole line from the SAM file and then parse/extract all the fields from it. Using

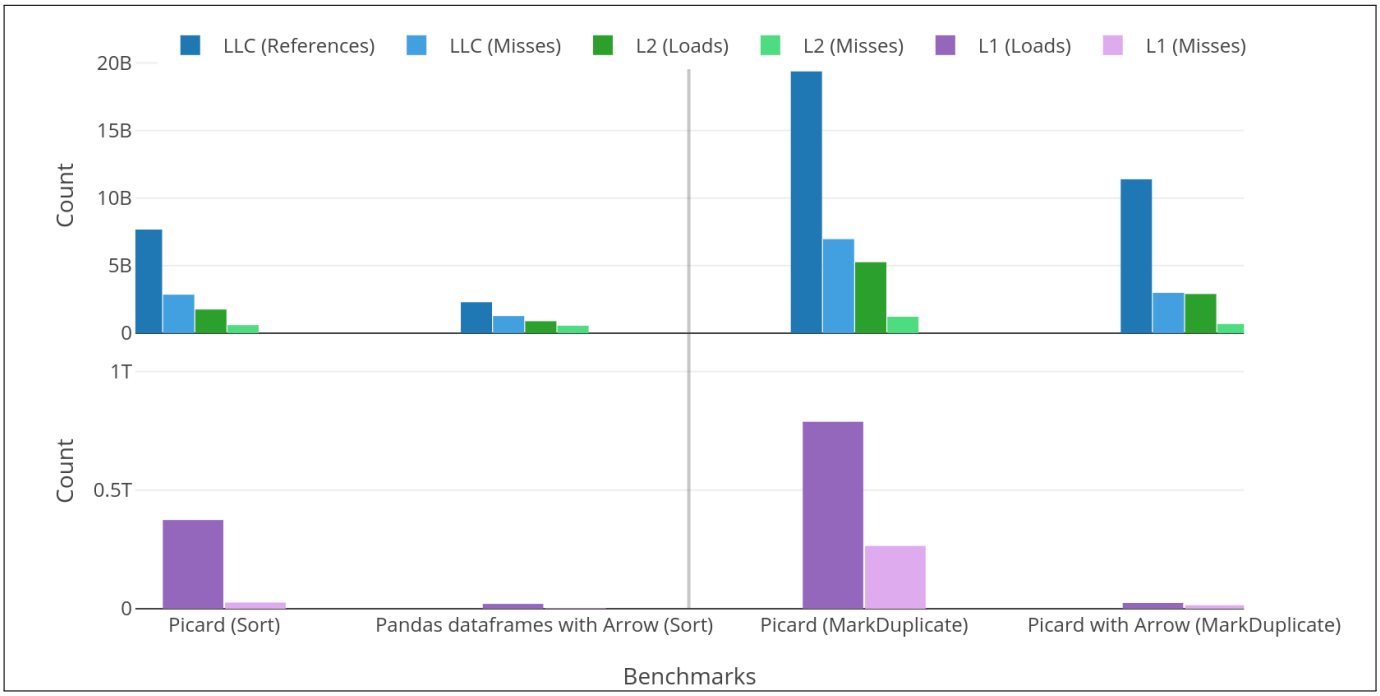

Fig. 6. A comparsion of cache L1, L2 and LLC statistics for Picard Sorting and MarkDuplicate applications with their respective ArrowSAM implementation for Whole Exome data set..

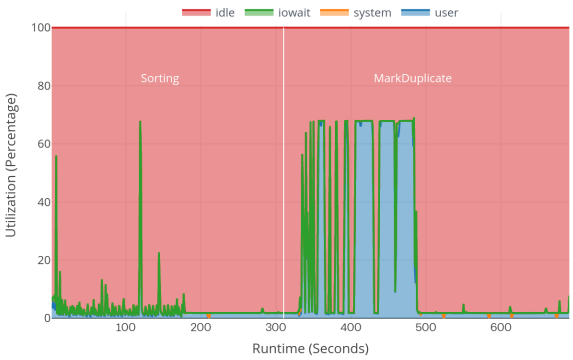

Fig. 7. CPU resources utilization summary for running Picard Sorting and MarkDuplicate for Whole Exome data set.

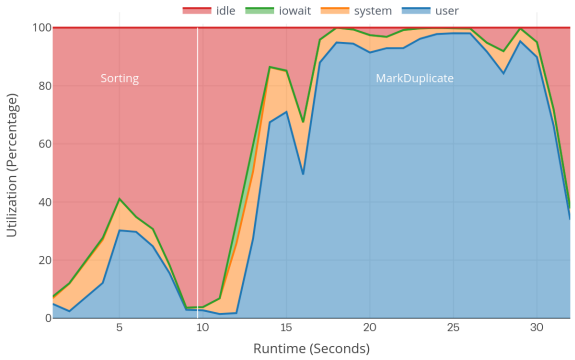

Fig. 8. CPU resources utilization summary for running Arrow based Picard Sorting and MarkDuplicate using ArrowSAM for Whole Exome data set.

Arrow in-memory columnar format we only access those SAM fields which are required by that specific tool to process.

3) Cache Locality: Due to Arrow in-memory columnar data format, our implementation is able to better exploit cache locality. As shown in Figure 6, all levels of cache accesses decrease due to fewer number of in-memory fields that need to be accessed for sorting and marking duplicate processes in WES. Cache miss rate also decreases in all cache levels and particularly in level-1 cache.

4) Memory Usage: ArrowSAM data representation is solely memory based. So all the data is placed in shared memory pool of Plasma Objects. After sorting the previous Plasma Objects can be removed to free the space to store new sorted data set which is used in next applications, other than this no additional memory is required for intermediate operations. Memory used by ArrowSAM and elPrep for WES and WGS data sets in pre-processing applications is shown in Table V.

TABLE V
MEMORY FOOTPRINT FOR IN-MEMORY PROCESSING TOOLS

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

VI. Related Work

Many in-memory implementations of genomic variant discovery pipeline have been presented previously, almost all

these implementations are cluster scaled and do not exploit the single node performance. A lot of them use the MapReduce programming model to distribute the jobs on cluster and some of them take the benefit of Apache Spark framework [22] for in-memory data management. Our focus is to exploit performance of single node systems.

elPrep [23], is a set of tools for pre-processing SAM/BAM files for variant calling. It's a multi-threaded execution engine which processes the data in memory instead of reading and writing to I/O for each of operations. Despite exploiting the parallelism and in-memory processing, the results show a maximum of 2-3x speedup achieved in all the benchmarks as the authors compared with different whole genome and exome data sets with single threaded Picard and SAMTools. In elPrep 4 [24], the authors reported 13x speedup over GATK best practices pipeline for whole-exome and 7.4x speedup for whole-genome data using maximum memory and storage footprints. They also compare the results on cluster to show that this platform is salable for high performance computing infrastructure and cost efficient. Memory-driven computing [25], in this research a huge pool of different types of memories created and connected to the processing resources through Gen-Z communication protocol. The memory is shared across are the 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.

VII. Conclusion and future work

This paper proposes a new in-memory SAM data representation that makes use of the in-memory capabilities of the Apache Arrow format. This format brings about 4 distinct benefits to genomics data access:

1) Data access can benefit from the standardized, cross-languages in-memory data representation of Arrow, which provides insight into the organization of the data sets through its schema.
2) It also keeps data in contiguous and columnar format to exploit cache spatial locality and SIMD vectorization capabilities.
3) For variable length data, Arrow uses offsets instead of lengths for data access, which in turn enables parallel access to such data any where in the memory pool.
4) Plasma Object Store provide a lightweight (de)serialization interface for in-memory stored objects.

We have also shown the potential of using in-memory Apache Arrow columnar format for genomic data storage and processing. In two use case examples, first we have implemented `Sorting` stage and secondly modified the `MarkDuplicate` stage to process data in-memory through shared memory Plasma Objects in parallel and efficiently. Further by employing chromosome level load balancing techniques can achieve more speed-up and can increase overall system throughput. We also plan to extend our implementations for complete variant calling pipeline. We also plan to use Apache Arrow in big data frameworks like Spark for cluster scale scalability of genomics applications.

Though in variant calling processes more ArrowSAM fields will be accessed and particularly for long reads, caches may become dirty more oftenly but still there exist an opportunity for parallel execution and cross-language interoperability.

Acknowledgment

Thanks to Peter van't Hof from Leiden University Medical Center (LUMC), Netherlands for fruitful discussions which motivated us to adopt the Apache Arrow Framework for genomics in-memory data representation and processing.

References

[1] I. H. G. S. Consortium, "Finishing the euchromatic sequence of the human genome," *Nature*, vol. 431, no. 7011, pp. 931–945, 2004. [Online]. Available: https://doi.org/10.1038/nature03001

[2] D. Gurdasani, M. S. Sandhu, T. Porter, M. O. Pollard, and A. J. Mentzer, "Long reads: their purpose and place," *Human Molecular Genetics*, vol. 27, no. R2, pp. R234–R241, 05 2018. [Online]. Available: https://doi.org/10.1093/hmg/ddy177

[3] J. Eid, A. Fehr *et al.*, "Real-time dna sequencing from single polymerase molecules," *Science*, vol. 323, no. 5910, pp. 133–138, 2009. [Online]. Available: https://science.sciencemag.org/content/323/5910/133

[4] H. . P. Wong, S. Raoux, S. Kim, J. Liang, J. P. Reifenberg, B. Rajendran, M. Asheghi, and K. E. Goodson, "Phase change memory," *Proceedings of the IEEE*, vol. 98, no. 12, pp. 2201–2227, Dec 2010.

[5] Y. Diao, A. Roy, and T. Bloom, "Building highly-optimized, low-latency pipelines for genomic data analysis."

[6] H. Li, "The sequence alignment/map format and samtools," *Bioinformatics*, vol. 25, pp. 2078 – 2079, 01 2009.

[7] "Picard toolkit," http://broadinstitute.github.io/picard/, 2019.

[8] A. Tarasov, A. J. Vilella, E. Cuppen, I. J. Nijman, and P. Prins, "Sambamba: fast processing of ngs alignment formats," *Bioinformatics*, vol. 31, no. 12, pp. 2032–2034, Jun 2015, 25697820[pmid]. [Online]. Available: https://www.ncbi.nlm.nih.gov/pubmed/25697820

[9] G. G. Faust and I. M. Hall, "Samblaster: fast duplicate marking and structural variant read extraction," *Bioinformatics*, vol. 30, no. 17, pp. 2503–2505, Sep 2014, 24812344[pmid]. [Online]. Available: https://www.ncbi.nlm.nih.gov/pubmed/24812344

[10] J. Goecks, A. Nekrutenko, J. Taylor, and T. G. Team, "Galaxy: a comprehensive approach for supporting accessible, reproducible, and transparent computational research in the life sciences," *Genome Biology*, vol. 11, no. 8, p. R86, 2010. [Online]. Available: https://doi.org/10.1186/gb-2010-11-8-r86

[11] D. Blankenberg, G. V. Kuster, N. Coraor, G. Ananda, R. Lazarus, M. Mangan, A. Nekrutenko, and J. Taylor, "Galaxy: A web-based genome analysis tool for experimentalists," *Current Protocols in Molecular Biology*, vol. 89, no. 1, pp. 19.10.1–19.10.21, 2010.

[12] S. Bridges. (2019) Seven bridges : Platform for genomic and healthcare data analytics. [Online]. Available: https://www.sevenbridges.com/

[13] GenomeNext. (2019) Genomenext: Intelligent translational medicine. [Online]. Available: https://www.genomenext.com/

[14] C. Chiang, R. M. Layer, G. G. Faust, M. R. Lindberg, D. B. Rose, E. P. Garrison, G. T. Marth, A. R. Quinlan, and I. M. Hall, "Speedseq: ultra-fast personal genome analysis and interpretation," *Nature Methods*, vol. 12, pp. 966 EP –, Aug 2015. [Online]. Available: https://doi.org/10.1038/nmeth.3505

[15] R. Valls Guimera, "Bcbio-nextgen: Automated, distributed, next-gen sequencing pipeline," *EMBnet.journal*, vol. 17, p. 30, 03 2012.

[16] J. L. Causey, C. Ashby, K. Walker, Z. P. Wang, M. Yang, Y. Guan, J. H. Moore, and X. Huang, "Dnap: A pipeline for dna-seq data analysis," *Scientific Reports*, vol. 8, no. 1, p. 6793, 2018. [Online]. Available: https://doi.org/10.1038/s41598-018-25022-6

[17] B. Institute. (2019) Introduction to the gatk best practices. [Online]. Available: https://software.broadinstitute.org/gatk/best-practices/

[18] Apache. (2019) Apache arrow: A cross-language development platform for in-memory data. [Online]. Available: https://arrow.apache.org/

[19] 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*, C. Hochberger, B. Nelson, A. Koch, R. Woods, and P. Diniz, Eds. Cham: Springer International Publishing, 2019, pp. 32–47.

[20] Apache. (2019) Plasma in-memory object store. [Online]. Available: https://arrow.apache.org/blog/2017/08/08/plasma-in-memory-object-store/

[21] L. Torvalds. (2019) perf: Linux profiling with performance counters.

[22] Apache. (2019) Apache spark: Lightning-fast unified analytics engine. [Online]. Available: https://spark.apache.org/

[23] C. Herzeel, P. Costanza, D. Decap, J. Fostier, and J. Reumers, "elPrep: High-performance preparation of sequence alignment/map files for variant calling," *PLOS ONE*, vol. 10, no. 7, p. e0132868, Jul. 2015. [Online]. Available: https://doi.org/10.1371/journal.pone.0132868

[24] C. Herzeel, P. Costanza, D. Decap, J. Fostier, and W. Verachtert, "elPrep 4: A multithreaded framework for sequence analysis," *PLOS ONE*, vol. 14, no. 2, p. e0209523, Feb. 2019. [Online]. Available: https://doi.org/10.1371/journal.pone.0209523

[25] 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," Jan. 2019. [Online]. Available: https://doi.org/10.1101/519579