Efficient Variant Calling on Human Genome Sequences Using a GPU-Enabled Commodity Cluster

Manas Jyoti Das
mjdbz4@health.missouri.edu
University of Missouri
USA

Khawar Shehzad
khawar.shehzad@mail.missouri.edu
University of Missouri
USA

Praveen Rao
praveen.rao@missouri.edu
University of Missouri
USA

ABSTRACT

Human genome sequences are very large in size and require significant compute and storage resources for processing and analysis. Variant calling is a key task performed on an individual's genome to identify different types of variants. Knowing these variants can lead to new advances in disease diagnosis and treatment. In this work, we propose a new approach for accelerating variant calling pipelines on a large workload of human genomes using a commodity cluster with graphics processing units (GPUs). Our approach has two salient features: First, it enables a pipeline stage to use GPUs and/or CPUs based on the availability of resources in the cluster. Second, it employs a mutual exclusion strategy for executing a pipeline stage on the GPUs of a cluster node so that the stages (for other sequences) can be executed using CPUs if needed. We evaluated our approach on a 8-node cluster with bare metal servers and virtual machines (VMs) containing different types of GPUs. On publicly available genome sequences, our approach was *3.6X-5X* faster compared to an approach that used only the cluster CPUs.

CCS CONCEPTS

• **Computing methodologies**; • **Applied computing**;

KEYWORDS

Variant calling, human genomes, cluster computing, GPUs

ACM Reference Format:
Manas Jyoti Das, Khawar Shehzad, and Praveen Rao. 2023. Efficient Variant Calling on Human Genome Sequences Using a GPU-Enabled Commodity Cluster. In *Proceedings of the 32nd ACM International Conference on Information and Knowledge Management (CIKM '23), October 21–25, 2023, Birmingham, United Kingdom.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3583780.3615268

1 INTRODUCTION

The volume of human genome data has continued to grow rapidly due to advances in sequencing technologies and lower cost of sequencing [28, 52]. A whole genome sequence of a human can consume gigabytes of storage space due to millions of reads [19, 54] produced by a sequencer [1]. These reads are short (overlapping) fragments of the deoxyribonucleic acid (DNA) in the genome. The analysis of a genome sequence begins by executing a *variant calling pipeline* [30], a key task to identify variants in the genome compared to a reference genome [35]. Several types of variants can be detected such as single nucleotide polymorphisms (SNPs), short insertions/deletions (indels), copy number variation, and so on [14]. Executing the pipeline is compute and I/O intensive as it involves reading the sequence data, aligning the reads against a reference genome, additional pre-processing steps to mitigate sequencing errors, and executing a variant caller to produce raw variants.

In recent years, there has been much interest in accelerating variant calling pipelines using parallel/distributed computing and hardware accelerators. Open source projects have emerged (e.g., GATK4 [27], ADAM/Cannoli [39, 40]) that employ cluster computing frameworks (e.g., Apache Spark [59], Apache Hadoop [55]) for variant calling on human genomes. Companies such as Microsoft, Google, NVIDIA, and Illumina are developing faster and more accurate solutions for human genome sequence analysis [9, 24, 34, 41, 45, 51, 58]. There is continued interest in advancing the state of the art for managing and analyzing human genomes at scale [53].

Recently, Rao et. al. proposed AVAH [47], an asynchronous computation model to improve cluster utilization of variant calling pipelines in a commodity cluster. AVAH achieved significant speedup compared to ADAM/Cannoli [39] by solely using CPUs. Today, GPUs are readily available in cloud and high performance computing (HPC) environments. Therefore, in this paper, we investigate the problem of *accelerating a variant calling pipeline on a large workload of human genomes using a GPU-enabled cluster.* Our key contributions are as follows:

• We propose a new approach that effectively manages both GPUs and CPUs in a cluster to accelerate a variant calling pipeline. Our approach is designed to achieve good utilization of the cluster resources via asynchronous computations.

• Our approach has two salient features: First, it enables a pipeline stage to use GPUs and/or CPUs based on the availability of resources in the cluster. Second, it employs a mutual exclusion strategy for executing a pipeline stage on the GPUs of a cluster node so that the stages (for other sequences) can use CPUs if needed.

• We evaluated our approach using a 8-node cluster with bare metal servers and VMs and different types of GPUs. We used the GATK4 [27] variant calling pipeline as it is widely adopted. On publicly available genome sequences, our GPU-aware approach was *3.6X-5X* faster than AVAH that used only the cluster CPUs.

The rest of the paper is organized as follows: Section 2 provides background/related work on variant calling pipelines. Section 3 presents our GPU-enabled approach. Section 4 describes the performance evaluation. Finally, we conclude in Section 5.

CIKM '23, October 21–25, 2023, Birmingham, United Kingdom

ACM ISBN 979-8-4007-0124-5/23/10…$15.00
https://doi.org/10.1145/3583780.3615268

2 BACKGROUND AND RELATED WORK

a) Accelerating Variant Calling Pipelines. Several efforts have been made to accelerate DNA variant calling pipelines using cluster computing frameworks. Some of them used Apache Hadoop [55] for speeding up the computationally-intensive alignment stage [2, 36, 37, 44, 48]. A few approaches used Apache Spark [59] for accelerating the alignment stage [3, 8, 60]. Even field-programmable gate arrays (FPGAs) were used to speed up alignment [4, 7] using BWA-MEM [32], a widely used alignment software. Additional tools were developed to cope with large genome datasets [31, 38, 50] and perform ad hoc analysis using Apache Hadoop and Pig [43]. However, only certain stages of a pipeline were supported.

The GATK Best Practices Workflows [26] are widely adopted for variant discovery. Halvade [11] parallelized the variant calling pipeline of GATK using MapReduce [10]. Later, GATK4 was released that employed Apache Spark for multithreading and parallelization [27]. NVIDIA developed Parabricks to accelerate GATK pipelines using GPUs [41, 42]. Google developed DeepVariant [45, 58] that used deep learning for variant calling and operated directly on aligned reads. Nothaft et. al. [39] created ADAM/Cannoli to handle large genomic datasets using Apache Spark and parallelized the alignment process/variant calling by reusing existing tools.

Yang et. al. [57] parallelized the code of DeepVariant [20] to run faster on GPUs. More recently, Illumina developed DRAGEN to accelerate the variant calling pipeline using FPGAs [23, 49]. Sentieon [51] developed highly optimized software-based algorithms for variant calling using CPUs. They developed DNAscope [15], a machine learning-based variant caller that achieved better accuracy than GATK HaplotypeCaller [25]. Recently, a few approaches used FPGAs to accelerate variant calling [33, 56].

b) AVAH [46, 47]. AVAH was proposed to overcome the poor cluster utilization of ADAM/Cannoli for variant calling on a workload of human genome sequences. AVAH improved the cluster utilization via the concept of futures [22], which enables non-blocking operations. AVAH distributed the task of executing a variant calling pipeline on input sequences across the cluster nodes. It exploited task parallelism and data parallelism for different stages in the pipeline via asynchronous computations (i.e., futures). These computations were executed in a sliding window manner on small groups of sequences resulting in improved cluster utilization. AVAH was built atop Apache Spark and Apache Hadoop. Hence, it leveraged the APIs of Adam/Cannoli [18, 39] for data parallelism. AVAH was 3X-4.7X faster than Adam/Cannoli on a large workload of low coverage human genome sequences [47].

Let us consider a single-sample (germline) variant calling pipeline with four stages [30] involving (a) reading FASTQ files [29] containing raw unmapped reads in preparation for alignment, (b) aligning reads [32] with a reference genome to produce mapped reads in the BAM format [21], (c) marking duplicate reads, BQSR and indel realignment to correct sequencing errors, and sorting, (d) invoking a variant caller [17, 25] to produce raw variants in the VCF format [16]. Algorithm 1 shows AVAH's strategy to process a workload of sequences for the aforementioned 4-stage variant calling pipeline. Each sequence has an ID, and the paired-end FASTQ files are stored in HDFS. The sequence IDs are read into Spark's resilient distributed dataset (RDD), which is partitioned across the worker nodes. AVAH invokes a map operation on each RDD partition to execute a stage on the sequences in that partition. It chains the map operations followed by *collect*.

Algorithm 1 Key steps in AVAH (a.k.a. AVAH$_y$) [47]

Input: f: a partitioning function; p: # of RDD partitions; ω: sliding window size for futures

1: Create an RDD R of sequence IDs, repartition into p partitions, and sort each partition by sequence size
2: $V \leftarrow R$.mapPartitions(executefutures(s_1, ω))
.mapPartitions(executefutures(s_2, ω))
.mapPartitions(executefutures(s_3, ω))
.mapPartitions(execFutures(s_4, ω)).collect()
3: **return** V // This .vcf file is stored in HDFS

c) Motivation. Most of the aforementioned techniques [24, 27, 39, 42, 45, 51] aim to accelerate the variant calling pipeline on a single *gold-standard* high coverage human genome sequence. However, our goal is to accelerate a large workload of human genome sequences using a commodity cluster similar to AVAH. The commoditization of GPUs provides an opportunity to outperform AVAH, which only uses the cluster CPUs. However, effectively managing the cluster GPUs/CPUs when executing different pipeline stages (of different sequences via futures) is a non-trivial challenge and requires rethinking how the stages are designed.

3 OUR APPROACH: AVAH*

In this section, we present AVAH* to accelerate a variant calling pipeline on a large workload of genomes using a GPU-enabled cluster. While AVAH* draws inspiration from AVAH in terms of asynchronous computations, it effectively manages the cluster CPU/GPUs to enable stages to use GPUs and/or CPUs.

Table 1: Stages in the GATK4 Pipeline

Stage	Description
s_1	Copy FASTQ files from HDFS to a local file system, produce an unaligned .bam, and store it in HDFS
s_2	Align the .bam file against a reference genome, mark duplicates of mapped reads to produce an aligned .bam file, and then store in HDFS
s_3	Sort the aligned reads and apply BQSR/indel realignment to produce a .bam file in HDFS
s_4	Invoke the variant caller (*HaplotypeCaller*) to produce a .vcf file in HDFS

Without loss of generality, we consider GATK4 [27] as the underlying variant calling pipeline as it is widely adopted. Table 1 shows the different stages of the pipeline. GATK4 supports multithreading and parallelization using Apache Spark. Hence, for the three stages s_2, s_3, and s_4, GATK4 can exploit data parallelism due to Spark-based APIs. However, for stage s_1, GATK4 still requires local processing on a cluster node as it currently does have a Spark-based implementation. Hence, s_1 requires more I/O due to copying FASTQ files from HDFS (to local storage) and copying back the unaligned .bam file (from local storage) to HDFS for a sequence.

The design of AVAH* has two salient features in order to effectively manage the CPUs/GPUs in a cluster: The *first* feature

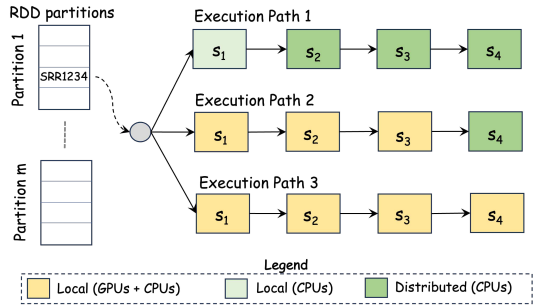


Figure 1: Possible Execution Paths in AVAH★

is its ability to execute a pipeline stage either on a single node's GPUs/CPUs or across multiple nodes using their CPUs. Figure 1 shows the possible execution paths for the pipeline stages. We use the term *local* to indicate that a single node's resources are used; we use the term *distributed* to indicate that the resources of several cluster nodes are used. *Execution Path 1* uses only CPUs to execute the stages shown in Table 1. *Execution Path 2*, however, uses local GPUs and CPUs (on a single node) to execute stages s_1-s_3, and finally, executes stage s_4 using the CPUs of several cluster nodes.[1] The *second* feature is a *mutual exclusion strategy* for executing a pipeline stage of a sequence on the GPUs of a single node. As a result, the stages of other sequences in the same RDD partition can either wait for the GPUs to become available or proceed to use CPUs. Without mutual exclusion, pipeline stages will fail due to limited memory on the GPUs.

Algorithm 2 Main steps for stage s_1

Input: *sid*: sequence ID; *useGPU*, *isFCFS*: Boolean flags
1: Copy FASTQ files for *sid* from HDFS to local file system
2: res ← `false`
3: Let *pID* denote the ID of the RDD partition
4: **if** useGPU = `true` AND (*pID* mod k = 0) **then**
5: res ← TryGPU(*isFCFS*, {s_1, s_2, s_3})
6: **if** res = `false` **then**
7: Construct .bam file using GATK4 on local FASTQ files
8: Copy unaligned .bam file (in s_1) to HDFS
9: res ← `true`
10: **return** (*sid*, res)

AVAH★ executes the pipeline on a workload of sequences using Algorithm 1. However, each stage s_1-s_4 that is executed as a *future* must be redesigned to be GPU-aware. We describe the main steps of these stages next. Algorithm 2 shows the steps for stage s_1. Based on the chosen positive integer k, either all or a subset of RDD partitions can be assigned to use GPUs using their IDs (Line 4). Within a partition, a first-come, first-served (FCFS) policy can be employed, wherein only the first future to acquire the lock proceeds to use GPUs. This is indicated by *isFCFS*=`true` (Line 5). Any other concurrent future that tries to acquire the same lock will return after a timeout period and proceed to use CPUs. If *isFCFS*=`false`, all the sequences in the partition will execute on the local GPUs one after the other. If *useGPU*=`false` (for *Execution Path 1*) or if GPUs were not available (for *Execution Paths 2 and 3*), the steps required by stage s_1 are executed locally on a node (Lines 7- 8).

[1]One can imagine another Execution Path where only s_1-s_2 use local GPU/CPU resources, and s_3-s_4 use distributed CPU resources. However, the current GPU-implementation of GATK4 [41] cannot execute only s_1-s_2 on the GPUs. Hence, this scenario is not considered as an Execution Path.

Algorithm 3 describes the mutual exclusion strategy for using GPUs that is invoked by stages s_1 and s_4 (to be discussed later). If *isFCFS*=`true` and the GPU lock is acquired by a future, the desired stages are executed on the node's GPUs. If the lock cannot be acquired, then the control is returned to caller. Otherwise, the process of acquiring the lock is tried repeatedly with some delay (Lines 8-10). Once the lock is acquired, the required stages are executed on the node's GPUs. Stages s_2 and s_3 are relatively straightforward and executed using distributed CPUs (Execution Path 1) if *useGPU*=`false`.

Algorithm 3 TryGPU: main steps for mutual exclusion

Input: *isFCFS*: Boolean flag; $\mathbb{S}$: stages to execute
1: lock *gpuLock* // one lock for all GPUs on a cluster node
2: *timeout* ← 30 seconds
3: **if** *isFCFS* = `true` **then**
4: **if** acquire(*gpuLock*, *timeout*) = `false` **then**
5: **return** `false`
6: **else**
7: res ← `false`
8: **while** res != `true` **do**
9: sleep("some seconds")
10: res ← acquire(*gpuLock*, *timeout*)
11: // The lock has been acquired
12: **for** each $s \in \mathbb{S}$ **do**
13: Perform stage s on the node's GPUs
14: Copy final output to HDFS // either .bam or .vcf
15: release(*gpuLock*)
16: **return** `true`

Algorithm 4 shows the steps for stage s_4 and is similar to Algorithm 2 in terms of assigning partitions to GPUs and mutual exclusion for GPU execution when *useGPU*=`true`. The distributed CPUs are used for s_4 when the GPU lock cannot be acquired for *isFCFS*=`true` or when *useGPU*=`false`.

Algorithm 4 Main steps for stage s_4

Input: *sid*: sequence ID; *useGPU*, *isFCFS*: Boolean flags
1: res ← `false`
2: Let *pID* denote the ID of the RDD partition
3: **if** useGPU = `true` AND (*pID* mod k = 0) **then**
4: res ← TryGPU(*isFCFS*, {s_4})
5: **if** res = `false` **then**
6: Perform stage s_4 using GATK4 Spark APIs
7: res ← `true`
8: **return** (*sid*, res)

4 PERFORMANCE EVALUATION

We begin by describing our experimental setup and then present the performance results of AVAH and AVAH★. Note that the variant calling pipeline/cluster hardware described below are different from those used in our previous work [47].

a) Dataset, Implementation and Cluster Setup. Our workload consisted of 98 human whole genome sequences that were publicly released by the 1000 Genomes Project [5]. The total size of these low-coverage (paired-end) sequences was 632 GB (in compressed form). The min. and max. size of the sequences (in compressed form) were 2.2 GB and 15.4 GB, respectively. They consumed 1.9 TB of HDFS storage due to the default replication factor of 3.

The original implementation of AVAH used the ADAM/Cannoli APIs [47]. Hence, we implemented support for GATK4 in AVAH

and the aforementioned algorithms of AVAH* using Scala 2.12.8 and GATK 4.1.8.0. In addition, Apache Spark 2.4.7, Apache Hadoop 2.7.6, OpenJDK 8, and CUDA Toolkit 11.0.2 were used. AVAH* used Parabricks 4.0.0 [41], which implements the GATK4 pipeline for GPUs via Docker containers [12]. The mutual exclusion strategy for GPUs was implemented using file locking in Linux.

We ran the experiments on a 8-node cluster that was set up in two different testbeds: CloudLab [13] and FABRIC [6]. CloudLab is an experimental testbed for cloud computing research and provides bare metal servers for experimentation. The cluster nodes ran Ubuntu 18.04 and were connected by a Gigabit Ethernet (10 Gbps). Each node had two Intel Xeon Silver 4114 10-core CPUs (2.2 GHz) and one NVIDIA GPU. Block storage was mounted on each node and used to set up HDFS. In contrast, FABRIC is a newer testbed with more powerful GPUs, faster storage drives, and high-speed optical links. FABRIC provides only VMs for experimentation. The VMs in the cluster ran Ubuntu 18.04 and were connected by a Gigabit Ethernet (25 Gbps). Non-volatile Memory Express (NVMe) drives were mounted on each VM. One NVIDIA GPU was attached to each VM. The physical servers that hosted the VMs had dual AMD EPYC Rome 2.5-2.9 GHz (32 core) processors with 512 GB RAM. Table 2 summarizes the cluster node/VM characteristics for the two testbeds. The GPU memory for P100, T4, and RTX6000 cards were 12 GB, 16 GB, and 24 GB, respectively.

Table 2: Hardware Comparison

Testbed	Logical cores/ node (or VM)	RAM/ node (or VM)	Storage/ node (or VM)	NVIDIA GPU type
CloudLab	40	192 GB	1.1 TB	P100
FABRIC	24	128 GB	850 GB	T4, RTX6000

Table 3: Time Comparison (Best Time Shown in Bold)

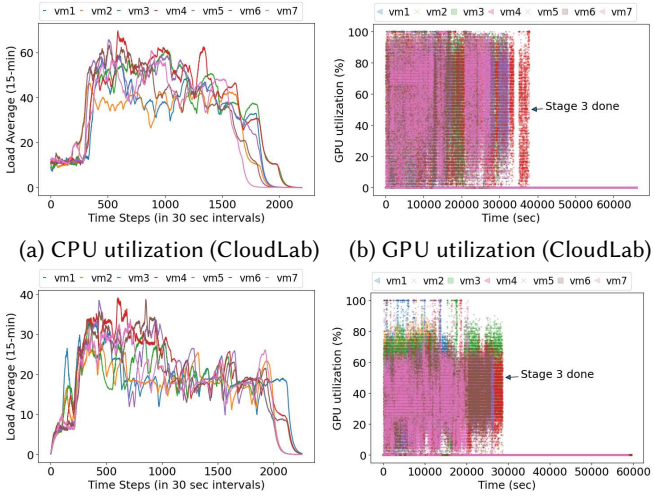
Testbed	Time taken			Best speedup
	AVAH	AVAH* (*isFCFS*=false)		
		$k = 1$	$k = 2$	
CloudLab	61 hr 30 m	**16 h 43 m**	35 h 52 m	3.67X
FABRIC	90 h 2 m	**17 h 49 m**	52 h 6 m	5.05X

b) Results. Next, we discuss the performance results. In each cluster setup, one node (or VM) was the *master* and the other nodes (or VMs) were the *workers*. AVAH and AVAH* were run using YARN [55] with the deploy mode as "client". Hence, all the Spark executors were launched on the worker nodes (or VMs). Thus, the master node (or VM) was lightly utilized as expected. Also, GATK's HaplotypeCaller failed to run on a P100 GPU due to insufficient memory. Hence, for all the experiments, AVAH* avoided using *Execution Path 3* for sequences. (Recall from Figure 1.) GRCh38 [35] was used as the reference genome. Both AVAH and AVAH* used a sliding window size of 2 for executing futures. We do not report any accuracy results as both AVAH and AVAH* use GATK4.

Table 3 shows the time taken by AVAH and some representative results for AVAH*. AVAH* ran the fastest for $k = 1$ and *isFCFS*=false, wherein every sequence was processed using *Execution Path 2*. In all cases, AVAH was the slowest as it used only CPUs and chose *Execution Path 1*. AVAH* outperformed AVAH in both the testbeds and achieved significant speedup by using GPUs.

Figures 2(a) and 2(b) show the overall CPU and GPU utilization of AVAH* in CloudLab, respectively. Figures 2(c) and 2(d) show the overall CPU and GPU utilization of AVAH* in FABRIC, respectively. The CPU utilization was measured using `dstat` and is reported as 15-minute load average (measured every 30 seconds) on a worker node/VM. The GPU utilization was measured using `nvidia-smi`. Overall, AVAH* achieved good CPU and GPU utilization. (Note that stage s_4 used (distributed) CPUs; hence, the GPU utilization was zero after stage s_3 completed for all the sequences.) In contrast, AVAH's cluster CPU utilization was lower than that of AVAH* in both testbeds. Hence, it ran slower than AVAH*. (In the interest of space, we do not show these plots.) In summary, AVAH*'s design of leveraging both CPUs and GPUs and the mutual exclusion strategy were effective in achieving good performance compared to solely using CPUs of a cluster. (We anticipate AVAH* to execute even faster if *Execution Path 3* was also used.)

Next, we remark on the execution of AVAH* in CloudLab and FABRIC. The cluster in CloudLab had slower storage drives. Hence, stage s_1 of AVAH* was more I/O bound on CloudLab compared to FABRIC, which had faster NVMe drives. We can observe this difference in Figures 2(a) and 2(c). With regard to GPUs, the cluster in FABRIC had more powerful GPUs (three T4 and four RTX6000 GPUs on the worker VMs) compared to CloudLab (seven P100 GPUs on worker nodes). Hence, the first three stages of the GATK4 pipeline finished much faster on FABRIC. This can be noticed in the GPU utilization plots shown in Figures 2(b) and 2(d).



(a) CPU utilization (CloudLab) (b) GPU utilization (CloudLab)

(c) CPU utilization (FABRIC) (d) GPU utilization (FABRIC)

Figure 2: Cluster Utilization of AVAH*

5 CONCLUSION

We presented AVAH* that leverages a GPU-enabled cluster to efficiently perform variant calling on human genomes using asynchronous computations and a mutual exclusion strategy. It was significantly faster than solely using cluster CPUs. AVAH* software is available at https://github.com/MU-Data-Science/GAF.

Acknowledgments. This work was supported by the National Science Foundation under Grant No. 2201583.

REFERENCES

[1] 2021. What is Paired-End Sequencing? https://www.illumina.com/science/technology/next-generation-sequencing/plan-experiments/paired-end-vs-single-read.html.

[2] J. M. Abuin, J. C. Pichel, T. F. Pena, and J. Amigo. 2015. BigBWA: Approaching the Burrows-Wheeler Aligner to Big Data Technologies. *Bioinformatics* 31, 24 (2015), 4003–4005.

[3] J. M. Abuin, J. C. Pichel, T. F. Pena, and J. Amigo. 2016. SparkBWA: Speeding up the Alignment of High-Throughput DNA Sequencing Data. *PLoS ONE* 11, 5 (2016).

[4] Nauman Ahmed, Vlad-Mihai Sima, Ernst Houtgast, Koen Bertels, and Zaid Al-Ars. 2015. Heterogeneous Hardware/Software Acceleration of the BWA-MEM DNA Alignment Algorithm. In *2015 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 240–246.

[5] Adam Auton and et.al. 2015. A Global Reference for Human Genetic Variation. *Nature* 526, 7571 (2015), 68–74.

[6] Ilya Baldin, Anita Nikolich, James Griffioen, Indermohan Inder S. Monga, Kuang-Ching Wang, Tom Lehman, and Paul Ruth. 2019. FABRIC: A National-Scale Programmable Experimental Network Infrastructure. *IEEE Internet Computing* 23, 6 (2019), 38–47.

[7] Yu-Ting Chen, Jason Cong, Zhenman Fang, Jie Lei, and Peng Wei. 2016. When Apache Spark Meets FPGAs: A Case Study for Next-Generation DNA Sequencing Acceleration. In *Proc. of the 8th USENIX Conference on Hot Topics in Cloud Computing* (Denver, CO). 64–70.

[8] J. Cong, Jie Lei, Sen Li, Myron Peto, P. Spellman, Peng Wei, and Peipei Zhou. 2015. CS-BWAMEM: A Fast and Scalable Read Aligner at the Cloud Scale for Whole Genome Sequencing. In *High Throughput Sequencing Algorithms and Applications (HITSEQ)*.

[9] Databricks. 2018. Building the Fastest DNASeq Pipeline at Scale. https://databricks.com/blog/2018/09/10/building-the-fastest-dnaseq-pipeline-at-scale.html

[10] Jeffrey Dean and Sanjay Ghemawat. 2004. MapReduce: Simplified Data Processing on Large Clusters. In *Proc. of the 6th OSDI Conference*. 137–150.

[11] D. Decap, J. Reumers, C. Herzeel, P. Costanza, and J. Fostier. 2015. Halvade: Scalable Sequence Analysis with MapReduce. *Bioinformatics* 31, 15 (2015), 2482–2488.

[12] Docker. 2023. Docker: Develop faster. Run anywhere. https://www.docker.com/

[13] Dmitry Duplyakin, Robert Ricci, Aleksander Maricq, Gary Wong, Jonathon Duerig, Eric Eide, Leigh Stoller, Mike Hibler, David Johnson, Kirk Webb, Aditya Akella, Kuangching Wang, Glenn Ricart, Larry Landweber, Chip Elliott, Michael Zink, Emmanuel Cecchet, Snigdhaswin Kar, and Prabodh Mishra. 2019. The Design and Operation of CloudLab. In *2019 USENIX Annual Technical Conference (USENIX ATC 19)* (Renton, WA). 1–14.

[14] Ensembl. 2021. Ensembl Variation - Variant Classification. https://m.ensembl.org/info/genome/variation/prediction/classification.html

[15] Donald Freed, Renke Pan, Haodong Chen, Zhipan Li, Jinnan Hu, and Rafael Aldana. 2022. DNAscope: High Accuracy Small Variant Calling Using Machine Learning. *bioRxiv* (2022). https://doi.org/10.1101/2022.05.20.492556

[16] GA4GH. 2021. The Variant Call Format (VCF) Version 4.2 Specification. https://samtools.github.io/hts-specs/VCFv4.2.pdf.

[17] Erik Garrison and Gabor Marth. 2012. Haplotype-Based Variant Detection from Short-Read Sequencing. arXiv:1207.3907

[18] Big Data Genomics. 2020. Big Data Genomics. https://github.com/bigdatagenomics/

[19] Sara Goodwin, John D McPherson, and W Richard McCombie. 2016. Coming of Age: Ten Years of Next-Generation Sequencing Technologies. *Nature Reviews Genetics* 17, 6 (2016), 333–351.

[20] Google. 2021. DeepVariant. https://github.com/google/deepvariant

[21] The SAM/BAM Format Specification Working Group. 2021. Sequence Alignment/Map Format Specification. https://samtools.github.io/hts-specs/SAMv1.pdf

[22] Robert H. Halstead. 1985. MULTILISP: A Language for Concurrent Symbolic Computation. *ACM TOPLAS* 7, 4 (1985), 501–538.

[23] Illumina. 2020. DRAGEN Wins at PrecisionFDA Truth Challenge V2 Showcase Accuracy Gains from Alt-aware Mapping and Graph Reference Genomes. https://www.illumina.com/science/genomics-research/articles/dragen-wins-precisionfda-challenge-accuracy-gains.html

[24] Illumina. 2022. DRAGEN. https://developer.illumina.com/dragen

[25] Broad Institute. 2020. HaplotypeCaller in a Nutshell. https://gatk.broadinstitute.org/hc/en-us/articles/360035531412-HaplotypeCaller-in-a-nutshell

[26] Broad Institute. 2021. Genome Analysis Toolkit. https://gatk.broadinstitute.org/hc/en-us

[27] Broad Institute. 2023. GATK4. https://gatk.broadinstitute.org/hc/en-us/articles/360035890591-Spark

[28] National Human Genome Research Institute. 2021. The Cost of Sequencing a Human Genome. https://www.genome.gov/about-genomics/fact-sheets/Sequencing-Human-Genome-cost

[29] Welcome Trust Sanger Institute. 2000. FASTQ Format Specification. https://maq.sourceforge.net/fastq.shtml

[30] Daniel C. Koboldt. 2020. Best Practices for Variant Calling in Clinical Sequencing. *Genome Medicine* 12, 1 (2020), 91.

[31] Simone Leo, Federico Santoni, and Gianluigi Zanetti. 2009. Biodoop: Bioinformatics on Hadoop. In *2009 International Conference on Parallel Processing Workshops*. IEEE, 415–422.

[32] Heng Li. 2013. Aligning Sequence Reads, Clone Sequences and Assembly Contigs With BWA-MEM. *arXiv e-prints* (March 2013), arXiv:1303.3997. arXiv:1303.3997

[33] Michael Lo, Zhenman Fang, Jie Wang, Peipei Zhou, Mau-Chung Frank Chang, and Jason Cong. 2020. Algorithm-Hardware Co-design for BQSR Acceleration in Genome Analysis ToolKit. In *2020 IEEE 28th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. 157–166.

[34] Microsoft. 2020. Microsoft Genomics. https://www.microsoft.com/en-us/genomics/

[35] NCBI. 2013. Genome Reference Consortium Human Build 38. https://www.ncbi.nlm.nih.gov/assembly/GCF_000001405.26/

[36] T. Nguyen, W. Shi, and D Ruden. 2011. CloudAligner: A Fast and Full-Featured MapReduce Based Tool for Sequence Mapping. *BMC Research Notes* 4, 1 (2011), 171.

[37] M. Niemenmaa, A. Kallio, A. Schumacher, P. Klemela, E. Korpelainen, and K. Heljanko. 2012. Hadoop-BAM: Directly Manipulating Next Generation Sequencing Data in the Cloud. *Bioinformatics* 28, 6 (2012), 876–877.

[38] Henrik Nordberg, Karan Bhatia, Kai Wang, and Zhong Wang. 2013. BioPig: a Hadoop-based Analytic Toolkit for Large-Scale Sequence Data. *Bioinformatics* 29, 23 (2013), 3014–3019.

[39] Frank A. Nothaft. 2017. *Scalable Systems and Algorithms for Genomic Variant Analysis*. Ph. D. Dissertation. UC Berkeley, ProQuest.

[40] Frank Austin Nothaft, Matt Massie, Timothy Danford, Zhao Zhang, Uri Laserson, Carl Yeksigian, Jey Kottalam, Arun Ahuja, Jeff Hammerbacher, Michael D. Linderman, Michael J. Franklin, Anthony D. Joseph, and David A. Patterson. 2015. Rethinking Data-Intensive Science Using Scalable Analytics Systems. In *Proc. of the 2015 ACM SIGMOD Conference* (Victoria, Australia). 631–646.

[41] NVIDIA. 2020. NVIDIA Clara Parabricks. https://developer.nvidia.com/clara-parabricks

[42] Kyle A. O'Connell, Zelaikha B. Yosufzai, Ross A. Campbell, Collin J. Lobb, Haley T. Engelken, Laura M. Gorrell, Thad B. Carlson, Josh J. Catana, Dina Mikdadi, Vivien R. Bonazzi, and Juergen A. Klenk. 2023. Accelerating Genomic Workflows Using NVIDIA Parabricks. *BMC Bioinformatics* 24 (2023).

[43] Christopher Olston, Benjamin Reed, Utkarsh Srivastava, Ravi Kumar, and Andrew Tomkins. 2008. Pig Latin: a Not-So-Foreign Language for Data Processing. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*. 1099–1110.

[44] Luca Pireddu, Simone Leo, and Gianluigi Zanetti. 2011. SEAL: A Distributed Short Read Mapping and Duplicate Removal Tool. *Bioinformatics* 27, 15 (2011), 2159–2160.

[45] Ryan Poplin, Pi-Chuan Chang, David Alexander, Scott Schwartz, Thomas Colthurst, Alexander Ku, Dan Newburger, Jojo Dijamco, Nam Nguyen, Pegah T Afshar, Sam S Gross, Lizzie Dorfman, Cory Y McLean, and Mark A DePristo. 2018. A universal SNP and Small-Indel Variant Caller Using Deep Neural Networks. *Nature Biotechnology* 36, 10 (2018), 983–987.

[46] Praveen Rao and Arun Zachariah. 2022. Enabling Large-Scale Human Genome Sequence Analysis on CloudLab. In *IEEE INFOCOM 2022 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. 1–2.

[47] Praveen Rao, Arun Zachariah, Deepthi Rao, Peter Tonellato, Wesley Warren, and Eduardo Simoes. 2021. Accelerating Variant Calling on Human Genomes Using a Commodity Cluster. In *Proc. of 30th ACM International Conference on Information and Knowledge Management (CIKM)*. 3388–3392.

[48] Michael C. Schatz. 2009. CloudBurst: Highly Sensitive Read Mapping with MapReduce. *Bioinformatics* 25, 11 (2009), 1363–1369.

[49] Konrad Scheffler, Severine Catreux, Taylor O'Connell, Heejoon Jo, Varun Jain, Theo Heyns, Jeffrey Yuan, Lisa Murray, James Han, and Rami Mehio. 2023. Somatic small-variant calling methods in Illumina DRAGEN™ Secondary Analysis. *bioRxiv* (2023). arXiv:2023.03.23.534011

[50] André Schumacher, Luca Pireddu, Matti Niemenmaa, Aleksi Kallio, Eija Korpelainen, Gianluigi Zanetti, and Keijo Heljanko. 2014. SeqPig: Simple and Scalable Scripting for Large Sequencing Data Sets in Hadoop. *Bioinformatics* 30, 1 (2014), 119–120.

[51] Sentieon. 2023. Enabling Precision Data for Precision Medicine. https://www.sentieon.com/

[52] Zachary D. Stephens, Skylar Y. Lee, Faraz Faghri, Roy H. Campbell, Chengxiang Zhai, Miles J. Efron, Ravishankar Iyer, Michael C. Schatz, Saurabh Sinha, and Gene E. Robinson. 2015. Big Data: Astronomical or Genomical? *PLOS Biology* 13, 7 (2015), 1–11.

[53] Tomoya Tanjo, Yosuke Kawai, Katsushi Tokunaga, Osamu Ogasawara, and Masao Nagasaki. 2021. Practical Guide for Managing Large-Scale Human Genome Data in Research. *Journal of Human Genetics* 66, 1 (2021), 39–52.

[54] Stanford University. 2023. Stanford Genomics. https://med.stanford.edu/sfgf/services/sequencing.html

[55] Tom White. 2009. *Hadoop: The Definitive Guide* (1st ed.). O'Reilly Media, Inc.
[56] Tiancheng Xu, Scott Rixner, and Alan L. Cox. 2023. An FPGA Accelerator for Genome Variant Calling. *ACM Transactions on Reconfigurable Technology and Systems* (May 2023), 1–20.
[57] Chih-Han Yang, Jhih-Wun Zeng, Cheng-Yueh Liu, and Shih-Hao Hung. 2020. Accelerating Variant Calling with Parallelized DeepVariant. In *Proceedings of the International Conference on Research in Adaptive and Convergent Systems* (Gwangju, Republic of Korea). 13–18.
[58] Taedong Yun, Helen Li, Pi-Chuan Chang, Michael F Lin, Andrew Carroll, and Cory Y McLean. 2021. Accurate, Scalable Cohort Variant Calls Using DeepVariant and GLnexus. *Bioinformatics* 36, 24 (01 2021), 5582–5589.
[59] Matei Zaharia, Mosharaf Chowdhury, Michael J. Franklin, Scott Shenker, and Ion Stoica. 2010. Spark: Cluster Computing with Working Sets. In *Proc. of the 2nd USENIX Conference on Hot Topics in Cloud Computing*. Boston, MA, 10–10.
[60] Lingqi Zhang, Cheng Liu, and Shoubin Dong. 2019. PipeMEM: A Framework to Speed Up BWA-MEM in Spark with Low Overhead. *Genes* 10, 11 (2019).