Efficient and Scalable Workflows for Genomic Analyses

Subho S. Banerjee, Arjun P. Athreya, Liudmila S. Mainzer, C. Victor Jongeneel,
Wen-Mei Hwu, Zbigniew T. Kalbarczyk, Ravishankar K. Iyer
{ssbaner2, athreya2, lmainzer, vjongene, w-hwu, kalbarcz, rkiyer} @illinois.edu

University of Illinois at Urbana-Champaign, USA.

ABSTRACT

Recent growth in the volume of DNA sequence data and the associated computational costs of extracting meaningful information warrant the need for efficient computational systems at scale. In this work, we propose the Illinois Genomics Execution Environment (IGen), a framework for efficient and scalable genome analyses. The design philosophy of IGen is based on algorithmic analysis and extensive measurements on compute- and data-intensive genomic analyses workflows (such as variant discovery and genotyping analysis) executed on high-performance and cloud computing infrastructures. IGen leverages the advantages of existing designs and proposes new software improvements to overcome the inefficiencies we observe in our measurements. Based on these composite improvements, we demonstrate that IGen is able to accelerate the alignment from 13.1 hours to 10.8 hours (1.2×) and the variant from 10.1 hours to 1.25 hours (8×) calling on a single node, and its modular design scales efficiently in a parallel computing environment.

CCS Concepts

•**Applied computing → Computational genomics;** *Bioinformatics;* •**Software and its engineering → Software performance;** *Data flow architectures;*

Keywords

Design; Measurement; Performance; Bioinformatics; Genomics

1. INTRODUCTION

The recent drop in genome sequencing costs [1] and the consequent proliferation of DNA sequencing technology have enabled large sequencing projects that are having a significant impact on several fields, from plant and animal biology to clinical therapeutics. Projects like the 1000 Genomes Project [2] and the Cancer Genome Atlas [3] have produced petabytes of genomic sequence data. Further, large-scale sequencing projects [4] underway today are expected to produce two orders of magnitude more raw sequencing data than what is available today. This unprecedented growth of genomics data and high computational costs for subsequent analytics makes a compelling need for efficient and scalable computing systems.

DIDC'16, June 01 2016, Kyoto, Japan
© 2016 Copyright held by the owner/author(s). Publication rights licensed to ACM. ISBN 978-1-4503-4352-7/16/06…$15.00
DOI: http://dx.doi.org/10.1145/2912152.2912156

In this work, we propose IGen (the Illinois Genomics Execution Environment), an efficient and scalable computational framework for genomic analysis. In particular, we focus on *variant calling and genotyping analysis*, an approach that identifies mutations in a sample genome with respect to a population baseline genome. We chose this analysis keeping in mind its significance in clinical settings for disease diagnostics of potentially several hundred patients a day. Like many other genomic analyses, variant calling is a workflow comprising a chain of several analysis tools. For example, the Broad Institute Best Practice Guidelines workflow [5, 6] comprises alignment (e.g., [7–10]), SNP (single nucleotide polymorphism) detection (e.g., [5, 11]), and SNP genotyping (e.g., [5, 12]).

Using a systems approach, our key contributions include:

1. Identifying the presence of only a few mathematical kernels (algorithmic patterns) with common algorithmic structures, across several popular bioinformatics tools.
2. A profiling based study of performance bottlenecks of popular tools used in variant calling workflows on high memory workstations, HPC and cloud infrastructures.
3. The design of IGen, a genomic data analytics framework which builds on contributions 1 & 2 above. IGen overcomes our observed inefficiencies (e.g., low CPU and disk-bandwidth utilization and the inability to scale to large number of processors) while still preserving the advantages of existing designs.

IGen's design can significantly improve the handling of input data from disk to memory and memory to CPU, streamlining and enhancing computation. The highlight of IGen's performance is as follows: it can accelerate alignment by 1.2× (from 13.1 hours to 10.8 hours) and variant calling by 8× (10.1 hours to 1.25 hours). It can also horizontally scale out on representative HPC systems (such as Blue Waters [13]) and cloud infrastructures with almost linear performance scaling.

2. THE VARIANT-CALLING WORKFLOW

In this section, we briefly describe the structure of a *variant-calling workflow*. Variant calling statistically infers differences between a sampled genome and a *reference genome* that represents the "baseline" genome for a population of the candidate species. This workflow takes as input high-throughput sequencing (HTS) data [14] of a sample and a reference genome and produces as output a list of mutations (single nucleotide polymorphisms (SNPs), small insertions and deletions (indels)), loci (the positions on the reference genome at which they occur), and the confidence the algorithm has in the correctness of the answer. The variant-calling workflow is the preliminary step of a large number of biological and medical analyses that look to establish relationships between *genotypic* and *phenotypic* traits.

The Broad Institute Best Practices Guidelines [5,6] (BPG) recommends the step-by-step process for performing variant-discovery analysis in HTS datasets. This process has become the *de facto* standard amongst biologists and bioinformaticians for computing variants. The best practices guidelines divides the entire variant-calling process into three constituent phases:

1. *Preprocessing*: Deals with correctly mapping reads into their most likely point of origin in the genome and eliminating systemic errors in the HTS data. This phase is divided into the following steps:
   (a) *Alignment* [7,10,15,16]: Maps each short-read in the input to a position in the reference genome that the read most likely came from.
   (b) *Marking Duplicates* [17–19]: Mitigates biases introduced by duplicate reads due to polymerase chain reaction (PCR) amplification or to optical duplication in the sequencing machine.
   (c) *Local Realignment* [18,19]: Corrects alignment positions at the site of known mutation in a population. These errors occur becaused the error models used by aligners penalize local alignments containing indels more than mismatches.
   (d) *Base Quality Score Recalibration* (BQSR) [20]: Identifies and corrects systemic errors (inherent to the sequencing machines) in the measurement of base quality scores.
2. *Variant Calling* [21–26]: Identifying mutations in an alignment in comparison with a reference genome.
3. *Filtration and Refinement* [27]: Filters false positives from the set of called variants.

While our measurements address the entire BPG pipeline, our design of IGen addresses the alignment (the most time-consuming) and variant-calling (the most algorithmically complex) aspects of the BPG pipeline.

3. PERFORMANCE PATHOLOGIES IN THE VARIANT-CALLING WORKFLOW

To design an efficient and scalable computational framework, we first wanted to learn where the existing computational infrastructures perform well and where they lack in performance. Further, we wanted to know the factors limiting performance so that such limitations could be addressed in our framework's design. In this section, we describe our comprehensive profiling study of the performance bottlenecks and limitations of the tools used in a variant-calling workflow. The experiments are performed on three popular BPG workflow configurations (Table 1) running on machines (Table 2) representative of an HPC supercomputer (Blue-Waters at NCSA, Illinois), a bare-metal node on a cloud infrastructure, and a high-memory workstation. Our input dataset for all test cases is a 60× coverage simulated [28] whole human genome with error models and read length distributions taken from an Illumina HiSeq 2500 sequencing machine. Based on our measurements (summarized in 3), we answer the following questions:

Question 1. What is the most efficient deployment strategy for the variant-calling workflow?

Question 2. What are the limiting factors in the performance of the tools used in the variant-calling workflow on today's premier computational platforms?

Question 3. How well does the variant-calling workflow lend itself to a "scale-out" execution model, such as cloud computing?

Workflow Stage	**Configurations of tools**		
	T1 [5, 6]	**T2 [29]**	**T3 [19, 26]**
Alignment	BWA	BWA	-
Mark Duplicates	Picard	Picard	ADAM
Realignment	GATK IndelRealigner	GATK IndelRealigner	ADAM
BQSR	GATK BaseRecalibrator	GATK BaseRecalibrator	ADAM
Variant Calling	GATK HaplotypeCaller	GATK HaplotypeCaller	Avocado

Table 1: Configurations of different variant-calling workflows for our measurement study. Configurations T1 and T2 correspond to single-node and distributed executions of the same tools, respectively.

System	**CPU**	**RAM**	**Storage**
Blue Waters (XE-6)	2×AMD Opteron 6300	64 GB	Lustre
Configuration 1 (C1)	2×Intel Xeon E5-2695v2 @ 2.40GHz	192 GB	Gluster over 100GbE; 800GB HDD
Configuration 2 (C2)	4×Intel Xeon E7-4860v2 @ 2.60GHz	3 TB	6×1 TB HDDs (software RAID 0), 4×8 TB SSDs (hardware RAID 0), RAMFS

Table 2: Configurations of test machines for measurement study

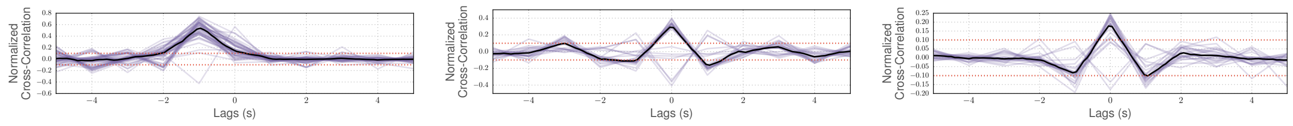
Question 1: Most efficient deployment strategy.

Table 3 shows the advantage of using the cloud-based ADAM and Avocado workflow in terms of overall performance (even though it does not perform alignment[1] alignment tool from the same authors would ensure faster processing than T2 or T3.). We see that tool-set T1 has the advantage of being deployable in a wide range of infrastructures. We also see that exploiting data-parallelism at the level of individual reads in the input dataset (as done in toolset T3) has a significant effect on performance as compared to coarse-grained data-parallelism (at the level of individual chromosomes) exploited in T2. Another factor to note is the of accuracy of the output. T1 (along with its distributed execution, T2) is the most widely accepted combination of tools whose answers are trusted by biologists and bioinformaticians.

[1]Using the SNAP [16]

Tool	**Platform**	**Machine**	**Walltime (hr)**	**Issues**
T1	Workstation (HDD)	C2	59.1	(1) Unnecessary and inefficient disk IO. (2) CPU bound by memory bandwidth. (3) Tools require linearly sorted data, which takes time to compute and hampers parallel processing. (4) GATK framework interface is insufficient to express any complex reduction tasks
	Workstation (RAMDISK)	C2	41.2	
	Cloud	C1	78.1	
	HPC	XE-6	59.3	
T2	HPC	XE-6^D	29.3	(1) Cannot exploit fine-grained data-parallelism even though available in the problem (2) POSIX-based tools cannot fully utilize these systems
T3	Cloud	C1^D	11.4	(1) Serialization and network transfer bottleneck (2) CPU bound by memory bandwidth (3) Analysis dataset size bound by available memory
Dragen coprocessor	Hardware Accelerated		0.31*	(1) Some model parameters are compiled into hardware and cannot be changed (2) Significantly longer development cycles (3) Rigorous validation of output pending

Table 3: Summary of profiling results (analyzing a human genome at $60\times$ coverage) in this work. Results marked with * are publicly reported performance numbers by the tools' authors. System configurations marked with $\square^D$ refer to distributed executions across multiple nodes.



(a) Samtools Index (b) GATK IndelRealigner (c) GATK HaplotypeCaller

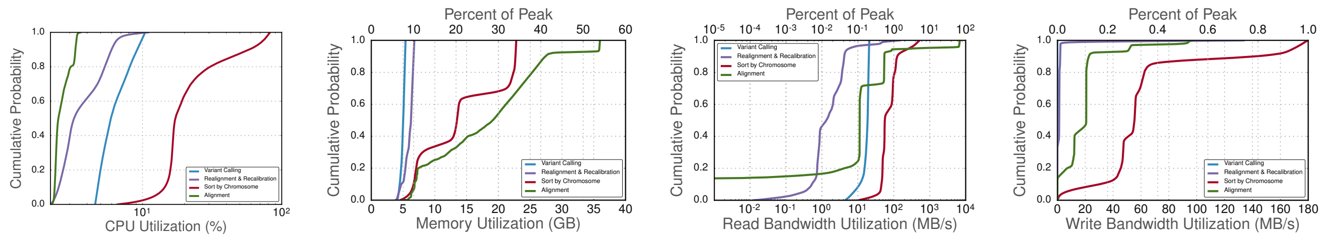
Figure 1: Normalized cross-correlation between IO and CPU utilization across the runtime of T1. The dark lines represent the median cross-correlation across 100 runs. The lighter lines represent the correlation coefficient for a single run. Red lines indicate a significance threshold.

There are recent advances in fully customized hardware for variant-calling workflows. Fully customized hardware solutions, such as Dragen, claim a runtime of the variant calling for the whole human genome of 0.31 hours (18 minutes), as shown in Table 3, while still closely following Broad Institute's best practices for variant calling. We clarify that comparing the runtime performance of custom hardware solutions with software solutions is not a fair one, and hence we will only compare our proposed framework's improvements with that of the existing software solutions. Further, the analytics algorithms of custom hardware solutions are not open-sourced, hence we do not speculate on their performance in comparable software implementations.

Question 2: Performance Limitations.

We observe that the primary performance bottleneck of the workflow is unnecessary and inefficient disk IO. The pathologies are as follows:

1. Large inputs ($\sim$ 100 GBs) are repeatedly read and written to disk between different stages of the workflow. Table 3 shows a $\sim$ 20% difference in performance between HDD and RAMDISK usage for T1 running on C2. T3 is able to avoid this issue by using in-memory computation.
2. Legacy data formats do not allow efficient random access. For example, a BAM file for storing alignments uses an R-Tree-based index to identify the mapping of reads to genome regions. However, a nontrivial amount of compute and data traversal has to be done to find all the nucleotides (from an alignment) at a particular locus in the reference.
3. Most steps of the workflow in T1 and T2 work by reading in a small amount of data from disk, computing on it, and then writing it back out. This means that individual read and write IOPs issued by the application are too small to utilize the entire disk bandwidth.
4. All IO operations are serialized (no asynchronous IO). Figure 1 shows the cross-correlation between IO and CPU utilization in some of the tools used in the workflow. By observing the peaks in IO and compute utilization, we see that the peaks are highly correlated and happen one after the other.
5. Inability to utilize a large parallel file system (common to HPC clusters). For example, configuration XE-6 uses a CRAY Sonexion Lustre file system. We observe (in Figure 2) that only a fraction of total bandwidth (theoretical peak 10 GB/s) is utilized. This can be attributed to: *a*) individual read and write IOPs being too small to use the entire bandwidth; and *b*) highly parallel file IO



(a) Distribution of CPU Utilization

(b) Distribution of Memory Utilization

(c) Distribution of IO Bandwidth to disk for reads

(d) Distribution of IO Bandwidth to disk for writes

Figure 2: Performance of the BPW Pipeline on the Blue-Waters supercomputer. Median resource utilization across CPU, memory, disk-read, and disk-write bandwidth is very low (approximately 10%).

(like that provided by MPI-IO) cannot be used under the POSIX interface for reads and writes.

Tool	**Walltime** (in hours)	**Bytes Read and Written**	**Instructions Executed**
BWA-MEM	13.10	9.53E11	1.38E15
Picard MarkDuplicates	7.211	9.81E11	4.84E14
Samtools SAM to BAM	8.11	8.39E11	7.61E14
GATK BaseRecalibrator	17.185	2.73E10	2.51E15
GATK RealignerTargetCreator	0.197	1.21E7	1.4E13
GATK IndelRealigner	6.499	3.51E10	2.48E14
GATK HaplotypeCaller	10.126	3.3E10	1.33E15

Table 4: Compute and disk IO in the primary steps of the BPW on C1 for a human genome @50× coverage (Note: OS buffer caches were cleared between executions of two tools.)

Table 4 shows the relation between runtime disk IO performed and computation performed at different stages of the workflow. It is apparent that almost all stages of the workflow are characterized by large volumes of data read and written. Some stages, like the GATK BaseRecalibrator, GATK IdelRealigner, and GATK HaplotypeCaller, are much more compute-intensive. In particular, we note:

1. The GATK framework provides an interface to traverse regions in an aligned genome through iterators called *walkers* (a sorted iterator over a specified region). This is a variant of the map-reduce paradigm but suffers because the API is insufficient to express many complex reduction tasks (e.g., random traversal of a region of the genome). As a result, all the tools based on GATK are constrained to traverse linearly sorted data. This hinders development of algorithms and tools that can efficiently make use of parallel computers.
2. While the GATK framework uses map-reduce as a programming abstraction, it does not use Map-Reduce as an execution strategy. This hinders it from using distributed execution runtimes like Hadoop [30] and Spark [31] to scale to large clusters.
3. Another significant source of inefficiency in the variant-calling pipeline is due to memory access patterns, particularly memory bandwidth constraints. For example, in the GATK tools used in the pipeline, we observe that ∼ 60% of the instructions executed are memory reads and writes. Of the remaining 40% instructions, stalls in the processor back end occur most commonly due to memory bandwidth limitations. Detailed results from our profiling experiments can be found in our technical report [32].
4. Poor design of reused data structures. For example, consider the following code snippet [2] in the HaplotypeCaller tool.

```
protected Map<GATKSAMRecord, Map<Allele,
↪   Double>> likelihoodReadMap = new
↪   LinkedHashMap<>();
```

Access to the `likelihoodReadMap` variable will incur several cache misses, and there is no way to predict any regular access pattens to this data structure. This problem (along with other, similar problems) has been noted by the developers [33] of GATK, but projects looking to fix these problems seemed to have stalled.

Question 3: Distributed Execution.

To address the throughput challenges of single-node execution of genomics pipelines, scientists apply techniques such as map-reduce [18, 26, 29, 30, 34, 35] and columnar storage [19, 36, 37]. This is reflected in the significant performance improvement in terms of distributing computation across multiple compute nodes (Toolsets T2 and T3 compared to T1 in Table 3).

Puckelwartz et al. [29] demonstrate that coarse-grained data parallelism (at the level of chromosomes) can utilize the advantages of a map-reduce style execution model for the workflow. We use a similar approach to distributing work across nodes on the Blue Waters supercomputer in T2. Figure 3 shows the distribution runtime of various tools in the BPG workflow executed on a single node (serial) and data-parallel version described above. We observe that the Realignment & Recalibration phase gains most from the data parallel execution and the alignment gains the least. This is because the data-parallel alignment phase has to merge

[2] https://github.com/broadgsa/gatk/blob/00fbcb36b2b2eba8a5c4ab32d26fa89c9637b04a/public/gatk-utils/src/main/java/org/broadinstitute/gatk/utils/genotyper/PerReadAlleleLikelihoodMap.java#L54

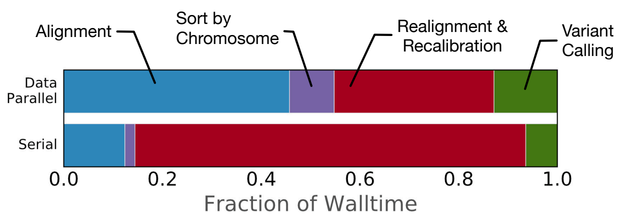


Figure 3: Comparing runtime of the BPG workflow for single node execution and data-parallel execution per chromosome

(reduce) the results of several parallel short-read aligner executions, results in an expensive on-disk sort of a couple of hundred GB of aligned short-reads. The exact runtime of the workflow can be computed by using the measurements in Figure 3 along with those for Configuration XE-6^D in Table 3. Overall, we observe that system resource utilization across the cluster (in Figure 2) for different phases of the workflow is comparable to the single-node execution and is quite poor. We believe this happens because: *a*) The application uses the file system as a distributed shared memory to cache data between executions. *b*) It is not able to take advantage of network based sorting techniques (like the map-reduce [30] shuffle). *c*) Underlying assumption of POSIX semantics for file systems means that these applications do not scale well. *d*) Coarse-grained data parallelism implies that the application can only scale to the number of chromosomes available in the input dataset, even though the problem affords a higher level of data parallelism. *e*) Load imbalance issues are caused by the distribution of sizes of chromosomes.

Similar approaches have been suggested [34,35] for the use of the Hadoop execution environment to parallelize the execution of tools in cloud environments by exploiting coarse-grained data parallelism. These approaches do achieve better performance (like Puckelwartz et al.), but they do not tackle the underlying performance pathologies associated with these bioinformatics tools. In general, they incur high overhead due to duplicate loading of indices and poor broadcasting of shared data.

Nothaft et al. [19,26] identify some of these problems with the BPG workflow for variant discovery. They demonstrate significant improvement in performance by using a columnar genomic data format ADAM and variant-caller Avocado running on top of Apache Spark. Our experiments with these tools (T3) on machine configuration C1 show that they perform significantly better (28 hours on 22 nodes vs. 11 hours on 4 nodes) than the T1 and T2 (see Table 3). This can be attributed to three major innovations in the ADAM-Avocado workflow: *a*) algorithmic changes that remove the requirement of sorted order of sequence data; *b*) use of columnar storage that provides a denser storage layout, thereby improving on-disk data layouts and efficient random access; and *c*) the ability to use hash-based partitioning and sorting using the network as provided by the underlying Spark data-shuffling engine. However, we still observe that IO is the biggest bottleneck in the system. In keeping with the use of an in-memory processing framework like Apache Spark, we see a significant decrease in disk IO use, however network IO between nodes increases significantly owing to the shuffling of data between nodes. We observe that the primary bottleneck in this process is the serialization/network-transfer/deserialization chain that must execute for all data shuffles. In Figure 4, we see that, 99% of the time, up to 14000 seconds (3.8 hours) was spent in serialization-deserialization, compression, and data transfer out of the 11.4 hour runtime observed in Table 3.

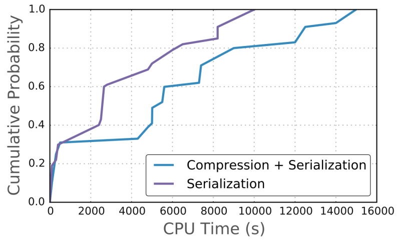


Figure 4: CDF showing the overall time spent in serializing, transmitting, and deserializing data in an execution of a whole human genome in the ADAM-Avocado framework

Algorithm 1 Basic algorithmic structure of a reassembly based variant caller

```
1: $alignment \leftarrow$ Aligned set of reads
2: $reference \leftarrow$ Reference genome for organism
3: $n \leftarrow$ Ploidy of the organism
4: $variants \leftarrow \emptyset$
5: $regions \leftarrow$ Find_Candidate_Regions($alignment, reference$)
6: for $region \in regions$ do
7:     $reads \leftarrow$ Reads_In_Region($region, alignment$)
8:     $assembly \leftarrow$ Generate_Assembly($reads$)
9:     $haplotypes \leftarrow$ Enumerate_Paths($assembly$)
10:    $realignment \leftarrow \emptyset$
11:    for $h \in haplotypes$ do
12:        for $r \in reads$ do
13:            $realignment[h] \leftarrow realignment[h] \cup$ Realign($h, r$)
14:        end for
15:    end for
16:    for $r \in reads$ do
17:        $h \leftarrow$ Best haplotype for $r$ in $realigment$
18:        Remove $r$ from $realignment[h']$ where $h' \neq h$
19:    end for
20:    for $h \in haplotypes$ do
21:        for $locus \in h$ do
22:            $variant \leftarrow variant \cup$ Genotype(Bases_At_Locus($locus, realignment[h]$))
23:        end for
24:    end for
25: end for
26: return $variants$
```

4. ALGORITHMIC PATTERNS

In this section, we present a study of a selection of algorithms used in a large number of genomic analyses (including variant calling, metagenomics, and phylogeny). Our study demonstrates:

1. The existence of common algorithmic *kernels* between different steps of the pipeline. The algorithmic kernels are a set of mathematical operations (e,g., Hidden Markov Models, computing edit distances) that are part of the larger set of computations in a tool. Several tools were manually inspected using their algorithm design and the code itself.

Analysis	Sub-Computation	Kernel Functions	Tools
Variant Calling	Alignment	Index computation, Needleman-Wunsch	SNAP, BWA, Bowtie
	Indel Realignment	Assembly, Edit-distance, Needleman-Wunsch	GATK IndelRealigner, ADAM
	BQSR	Yates Correction	GATK BaseRecalibrator, ADAM
	Variant Calling	Entropy, Convolution, Assembly, Edit-distance, Pair-HMM, Bayesian inference	GATK HaplotypeCaller, Avocado, Platypus
Metagenomics	Assembly	De-Bruijn Graph Construction	GeneStitch, MetaVelvet, Meta-IDBA, IDBA-UD
	Gene Calling	Inference on HMM	Genemark.hmm
	Species Diversity	Graph-Traversal, Inference on HMM	MEGAN, CARMA, Phymm
Phylogeny	Distance Method	Arithmetician, Minimization of L2-norm	PHYLIP, MEGA, TNT, ClustalW, PAUP, T-Rex, DARWIN
	Probabilistic Method	Inference on HMM	Clustal-Omega, HMMER, PFAM, UPP, TIPP, SEPP

Table 5: Summary of algorithmic kernels used across a variety of tools in variant calling, metagenomics, and phylogeny

2. Different bioinformatics tools that perform the same computation have very similar algorithmic structures and differ primarily in a few *kernels*.

This study encompasses a partial but significant number of tools used for a wide range of computational genomics analyses.

The existence of these patterns can be attributed to the mathematical models underlying the computation. For example, if two tools study similar biological phenomena, the underlying operations (computations) performed on the respective mathematical models are also similar. This observation gives us a natural partitioning of the larger algorithms into smaller tasks, as well as identifying dependencies between tasks. These can be described as data and control flow graphs (CDFG). For example, Figure 5(a) demonstrates one such graph for the GATK HaplotypeCaller. This allows us to reason about:

1. Available parallelism in the problem.
2. System level optimizations for a large number of bioinformatics tools:
   - (a) algorithmic kernels can be individually accelerated in hardware;
   - (b) accelerating reused kernels can target multiple tools at the same time; and
   - (c) runtime systems can be used to manage kernel executions and data movement across a large distributed system.
3. CDFGs can be easily and efficiently composed to form a workflow.

Table 5 summarizes this analysis of the algorithms used in variant calling, metagenomics, and phylogeny. A detailed analysis of the tools is included in our technical report [32, 38]. The generic algorithmic structure for reassembly-based variant calling and genotyping algorithms like GATK HaplotypeCaller [5,18], Platypus [24] and Avocado [25] is shown in Algorithm 1. Each of these tools defines a set of kernel functions to replace the generic ones found in the psuedocode of Algorithm 1. For example, Table 6 demonstrates the mathematical formulations of these functions for the GATK HaplotypeCaller tool.

Kernel Function	Mathematical Formulation
`Find_Candidate _Regions`	(1) Entropy calculation for each locus in the alignment; (2) Gaussian low-pass filtering of calculated entropy; (3) Thresholding activity profile to find candidate regions
`Generate_Assembly`	De-Bruijn Graph Assembly [39]
`Realign`	Pair-Hidden Markov Model [40]
`Genotype`	Statistical model defined for `mpileup` [22, 27]

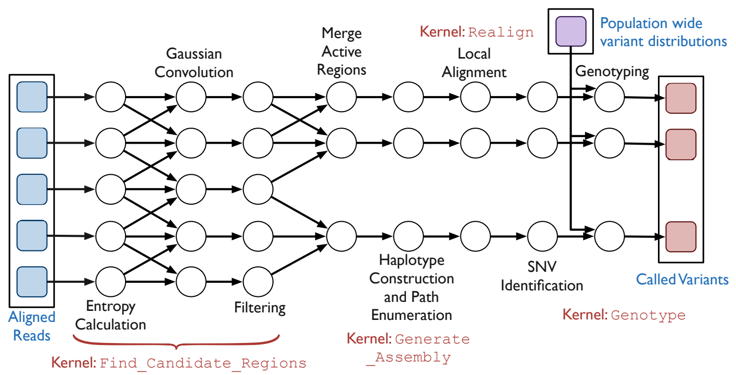
Table 6: Kernel functions involved in the GATK HaplotypeCaller as shown in Algorithm 1

5. THE ILLINOIS GENOMICS EXECUTION ENVIRONMENT (IGEN)

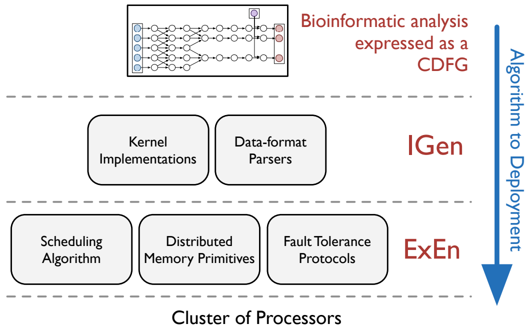
IGen is a framework for efficiently describing analytics that deal with NGS sequence data for high-performance execution on a single machine as well as scaling out to a cluster of nodes. It makes use of our observations from the Sections 3 and 4. IGen provides computational and IO primitives that can be used together to build genome analysis tools.

Algorithmic Patterns and Workflow Parallelization.

Based on our observation that several bioinformatics tools can be expressed as data-flow graphs composed of a small number of kernel functions, we designed the IGen framework as a library of kernel functions that can be used together to



(a) GATK HaplotypeCaller algorithm expressed as data-flow graphs



(b) Deployment strategy for a data-flow graph under the ExEn and IGen frameworks

Figure 5: The GATK HaplotypeCaller algorithm expressed as a CDFG using the IGen framework and its deployment strategy over a cluster using the ExEn framework

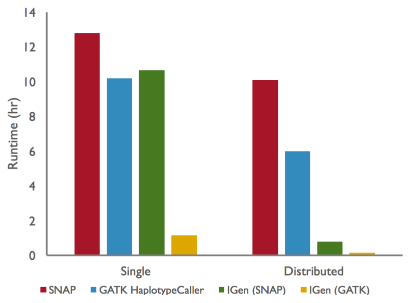


Figure 6: Comparing the runtimes of traditional tools with IGen-based tools

form meaningful bioinfomatics tools. Underlying IGen is a runtime library called ExEn for executing data-flow graphs and coordinating IO operations. Figure 5(b) shows the services provided by IGen and ExEn for taking an algorithm expressed as a CDFG of kernels and deploying it on a set of processors in a scalable and efficient manner. A detailed description of the ExEn runtime environment can be found in our technical report [32]. This approach embodies two key improvements over previous work:

1. *Optimizing performance of kernel functions.* IGen builds a list of kernels from Table 5 (currently only for the SNAP aligner and the GATK HaplotypeCaller for variant calling) that are able to make best use of modern processors through the use of SIMD and threading, by choosing between the openly available implementations of these functions and our own implementations.
2. *Data parallelism at several levels.* Our data-flow-graph-based representation of these bioinformatics tools can highlight the data parallelism available in the problem area at the level of individual NGS reads. The ExEn framework then uses this information to adaptively decide the granularity of the work it dispatches to a particular processor.

Improved Data Handling.

Our profiling measurements show that a number of data cleaning and quality control steps are IO intensive. Simply parallelizing these steps using more nodes will not resolve the problem that the CPUs are mostly idle due to outstanding IO activities. IGen uses a number of techniques to effectively perform IO:

1. *In-memory data layout.* IGen uses a Structure of Array (SoA) data layout to replace the widely used Array of Structures (AoS) layout for storing NGS data in memory. For example, a SAM file for aligned reads [21] is a collection of records stored consecutively, where each record for an aligned read contains 11 mandatory fields (e.g., read sequence, quality scores of its bases, the aligned position, etc.). Traditional tools have the drawback of having to iterate over record separators to identify records and then use a gather-scatter-based mechanism for memory access. The SoA layout allows for contiguous memory access patterns, allowing better locality and bandwidth utilization and reducing the need for CPU-Memory IO, and improved SIMD efficiency.
2. *On-disk data layouts (file formats).* In IGen, disk data layouts are bit-wise identical to the in-memory layouts described earlier. This removes the overhead of parsing input files from disk. As a result, all in-memory data structures that use pointers are constrained to be relative to the start of the buffer (dereferencing operations include an extra addition of a base pointer). Orthogonally, this gives us the ability to use asynchronous IO by mapping a file (through the `mmap` system call) to memory using the operating system. This solves the issues we observe in the ADAM file formats, where serialization and deserialization of data takes a significant portion of the runtime.
3. *Data representation.* We observe that all tools in the BPG workflow use a redundant 8-bit representation for nucleotides and quality scores. Instead, IGen uses a 3-bit representation for nucleotides (i.e., `A`, `C`, `G`, `T` and `N`), a 2-bit representation for CIGAR strings, and a 4-bit representation for quality scores. We observe that this change, coupled with the SoA data formats, provides a significant performance improvement, as all inputs now have a very compact and dense representation. This also improves overall bandwidth utilization and cache performance. Other work, such as CRAM [41], has performed compression on columnar representation of alignment data to improve overall performance by limiting IO.

In contrast, the approach used in IGen has the added advantage of not imposing additional restrictions on the ordering and structuring of the data.

Scalable Distributed Execution.

The ExEn runtime environment can use the one-sided communication available in MPI to allow distributed execution across a cluster of computers. The use of one-sided communication follows from our observations of time spent performing the serialization and deserialization of network data in ADAM and Avocado. One-sided communication fits well with our data model described above and allows us to use data transfers between nodes with almost no time spent in serialization. In addition, ExEn and IGen utilize network-based sorting and aggregation techniques to allow some phases of the computation to have a linearly sorted view of the data. This is a significant improvement in performance (for HPC machines) over the traditional disk-based sorting used in T1 and T2.

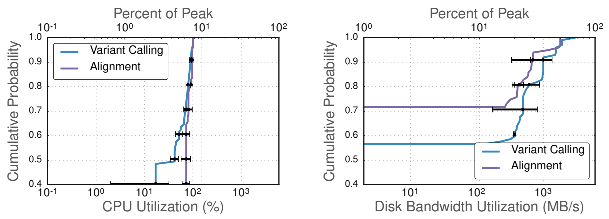


Figure 7: Distribution of CPU an disk IO bandwidth utilization for the alignment and variant-calling tools implemented in IGen

5.1 Results

We implement the SNAP alignment algorithm [16] and the GATK HaplotypeCaller algorithm in the IGen framework. We use a 60× coverage simulated human genome sample (same as our previously described measurements) to compare the performance of the original tools to those implemented in IGen. We run several experiments on the Blue-Waters XE-6 nodes (both for single node and distributed processing experiments). Figure 6 demonstrates the improvement in performance of the IGen-based implementation of the alignment and variant-calling tools compared to their original implementations. The alignment computation is accelerated by $\sim 1.2\times$ for a single XE-6 node (reduction from 13.1 hours to 10.8 hours) and by $\sim 13\times$ for a cluster of 10 XE-6 nodes. The variant-calling computation is accelerated by $\sim 8\times$ on a single XE-6 node (reduction from 10.1 hours to 1.25 hours) and by $\sim 70\times$ on a cluster of 10 XE-6 nodes. The alignment computation demonstrates super-linear scaling, as the in-network sorting of the output alignment is more efficient than the in-memory sort (and largely parallel). The HaplotypeCaller on the other hand does not scale well past 10 nodes. We believe this is because of the inherent nonuniform distribution of mutations in a genome.

Figure 7 demonstrates the overall CPU and disk bandwidth utilization of the IGen-based tools. Compared to Figure 2, we see that the our improvements to the data layouts and representation have led to a significantly higher median utilization of compute resources. Under IGen, we observe that the peak disk bandwidth utilized is much higher than previously ($\sim$ 1 GB/s compared to 10 MB/s for the HaplotypeCaller). However, we also see some load imbalance issues that contribute to significant variance in utilization seen across multiple experiments. Also, we notice that using `mmap` for mapping the same file to multiple machines on the network can lead to poor IO utilization. We believe this is because concurrent writes into a commonly shared file require synchronization across all nodes that have the file mapped in memory. We plan to address these issues in future work.

Tool	**TPR**	**TPR Confidence Interval** (95%)	**FPR**	**FPR Confidence Interval** (95%)
GATK Haplotype Caller	99.1	0.6	2.1×10^{-4}	0.13×10^{-5}
IGen	99.03	0.7	2.31×10^{-4}	0.06×10^{-5}

Table 7: Statistical validation of the correctness of the HaplotypeCaller algorithm implemented in IGen

In addition to the performance of these tools, our experiments test the correctness of the IGen-based implementations. Our experiments use simulated data (with injected mutations); hence the correct answers for alignments and called variants are known to us ahead of time. We check the answers of our implementation of the alignment algorithm by doing a bit-wise comparison (of alignment index and the CIGAR string) of the answers from IGen and SNAP. The validation of the variant-calling algorithm is more involved, as the output is dependant on the generation of a random variable. We statistically test the correctness of the answer by computing confidence intervals for the true positive and false positive rates of the HaplotypeCaller and IGen implementations. In Table 7, we see that both of these intervals have a significant overlap.

6. CONCLUSION

In this paper we presented the design of IGen, a framework for efficient and scalable genomic processing. We provided the motivation for our design through measurements that revealed bottlenecks in current variant-calling and genotyping workflow and through a study of algorithms that showed the presence of common mathematical *kernels* across a variety of analyses. Finally, we demonstrated that by addressing these performance bottlenecks, we can significantly improve the performance of the variant-calling workflow.

7. ACKNOWLEDGMENTS

This research was supported by several grants: in part by the National Science Foundation under Grant No. CNS 13-37732; in part by the Blue Waters sustained-petascale computing project supported by the National Science Foundation (awards OCI-0725070 and ACI-1238993) and the state of Illinois; and in part by IBM Faculty Awards to Profs. Hwu and Iyer. We gratefully acknowledge the support of Dr. Volodymyr Kindratenko and the Innovative Systems Laboratory at NCSA for facilitating our profiling experiments.

8. REFERENCES

[1] NHGRI. DNA sequencing costs. http://www.genome.gov/sequencingcosts/.

[2] Nayanah Siva. 1000 Genomes project. *Nature Biotechnology*, 26(3):256, March 2008.

[3] The Cancer Genome Atlas Research Network, John N. Weinstein, Eric A. Collisson, Gordon B. Mills, Kenna R. Shaw, Brad A. Ozenberger, Kyle Ellrott, Ilya Shmulevich, Chris Sander, and Joshua M. Stuart. The Cancer Genome Atlas Pan-Cancer Analysis Project. *Nature Genetics*, 45(10):1113–1120, September 2013.

[4] Genomics England. 100,000 genomes project. https://www.genomicsengland.co.uk/.

[5] Mark A. DePristo, Eric Banks, Ryan Poplin, Kiran V. Garimella, Jared R. Maguire, Christopher Hartl, Anthony A. Philippakis, Guillermo del Angel, Manuel A. Rivas, Matt Hanna, Aaron McKenna, Tim J. Fennell, Andrew M. Kernytsky, Andrey Y. Sivachenko, Kristian Cibulskis, Stacey B. Gabriel, David Altshuler, and Mark J. Daly. A framework for variation discovery and genotyping using next-generation dna sequencing data. *Nat Genet*, 43(5):491–498, May 2011.

[6] Geraldine A. Van der Auwera, Mauricio O. Carneiro, Christopher Hartl, Ryan Poplin, Guillermo del Angel, Ami Levy-Moonshine, Tadeusz Jordan, Khalid Shakir, David Roazen, Joel Thibault, Eric Banks, Kiran V. Garimella, David Altshuler, Stacey Gabriel, and Mark A. DePristo. *From FastQ Data to High-Confidence Variant Calls: The Genome Analysis Toolkit Best Practices Pipeline*. John Wiley & Sons, Inc., 2013.

[7] Ben Langmead, Cole Trapnell, Mihai Pop, and Steven L Salzberg. Ultrafast and memory-efficient alignment of short DNA sequences to the human genome. *Genome Biol*, 10(3):R25, 2009.

[8] H. Li and R. Durbin. Fast and accurate short-read alignment with burrows-wheeler rransform. *Bioinformatics*, 25(14):1754–1760, may 2009.

[9] H. Li and R. Durbin. Fast and accurate long-read alignment with burrows-wheeler rransform. *Bioinformatics*, 26(5):589–595, jan 2010.

[10] Yinan Li, Jignesh M. Patel, and Allison Terrell. Wham: A high-throughput sequence alignment method. *ACM Trans. Database Syst.*, 37(4):28:1–28:39, December 2012.

[11] R. Li, Y. Li, X. Fang, H. Yang, J. Wang, K. Kristiansen, and J. Wang. SNP detection for massively parallel whole-genome resequencing. *Genome Research*, 19(6):1124–1132, may 2009.

[12] Richard A Gibbs, John W Belmont, Paul Hardenbol, Thomas D Willis, Fuli Yu, Huanming Yang, Lan-Yang Ch'ang, Wei Huang, Bin Liu, Yan Shen, et al. The international hapmap project. *Nature*, 426(6968):789–796, 2003.

[13] University of Illinois at Urbana-Champaign National Center for Supercomputing Applications. Blue waters. http://www.ncsa.illinois.edu/enabling/bluewaters. Accessed: 2016-02-16.

[14] Michael L. Metzker. Sequencing technologies – the next generation. *Nat Rev Genet*, 11(1):31–46, Jan 2010.

[15] Heng Li and Richard Durbin. Fast and accurate long-read alignment with Burrows–Wheeler transform. *Bioinformatics*, 26(5):589–595, 2010.

[16] Matei Zaharia, William J Bolosky, Kristal Curtis, Armando Fox, David Patterson, Scott Shenker, Ion Stoica, Richard M Karp, and Taylor Sittler. Faster and more accurate sequence alignment with SNAP. *arXiv preprint arXiv:1111.5572*, 2011.

[17] The Broad Institute of Harvard and MIT. Picard. http://broadinstitute.github.io/picard/. Accessed: 2015-10-10.

[18] 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. *Genome Research*, 20(9):1297–1303, jul 2010.

[19] Frank Austin Nothaft, Matt Massie, Timothy Danford, Zhao Zhang, Uri Laserson, Carl Yeksigian, Jey Kottalam, Arun Ahuja, Jeff Hammerbacher, Michael Linderman, Michael J. Franklin, Anthony D. Joseph, and David A. Patterson. Rethinking data-intensive science using scalable analytics systems. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, SIGMOD '15, pages 631–646, New York, NY, USA, 2015. ACM.

[20] K. Nakamura, T. Oshima, T. Morimoto, S. Ikeda, H. Yoshikawa, Y. Shiwa, S. Ishikawa, M. C. Linak, A. Hirai, H. Takahashi, M. Altaf-Ul-Amin, N. Ogasawara, and S. Kanaya. Sequence-specific error profile of illumina sequencers. *Nucleic Acids Research*, 39(13):e90–e90, may 2011.

[21] Heng Li, Bob Handsaker, Alec Wysoker, Tim Fennell, Jue Ruan, Nils Homer, Gabor Marth, Goncalo Abecasis, Richard Durbin, et al. The sequence alignment/map format and SAMtools. *Bioinformatics*, 25(16):2078–2079, 2009.

[22] Heng Li. A statistical framework for SNP calling, mutation discovery, association mapping and population genetical parameter estimation from sequencing data. *Bioinformatics*, 27(21):2987–2993, 2011.

[23] Erik Garrison and Gabor Marth. Haplotype-based variant detection from short-read sequencing. *arXiv preprint arXiv:1207.3907*, 2012.

[24] Andy Rimmer, Hang Phan, Iain Mathieson, Zamin Iqbal, Stephen RF Twigg, Andrew OM Wilkie, Gil McVean, Gerton Lunter, WGS500 Consortium, et al. Integrating mapping-, assembly-and haplotype-based approaches for calling variants in clinical sequencing applications. *Nature Genetics*, 46(8):912–918, 2014.

[25] Big Data Genomics. Avocado. https://github.com/bigdatagenomics/avocado. Accessed: 2015-10-10.

[26] Frank Nothaft. Scalable genome resequencing with adam and avocado. Master's thesis, EECS Department, University of California, Berkeley, May 2015.

[27] H. Li. Toward better understanding of artifacts in variant calling from high-coverage samples. *Bioinformatics*, 30(20):2843–2851, jun 2014.

[28] Zachary Daniel Stephens. Empirical accuracy bounds for next-generation sequencing variant calling workflows. Master's thesis, University of Illinois at Urbana-Champaign, Urbana, IL, USA, 2015.

[29] M. J. Puckelwartz, L. L. Pesce, V. Nelakuditi, L. Dellefave-Castillo, J. R. Golbus, S. M. Day, T. P. Cappola, G. W. Dorn, I. T. Foster, and E. M. McNally. Supercomputing for the parallelization of whole genome analysis. *Bioinformatics*, 30(11):1508–1513, feb 2014.

[30] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: Simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113, January 2008.

[31] Matei Zaharia, Mosharaf Chowdhury, Michael J. Franklin, Scott Shenker, and Ion Stoica. Spark: Cluster computing with working sets. In *Proceedings of the 2Nd USENIX Conference on Hot Topics in Cloud Computing*, HotCloud'10, pages 10–10, Berkeley, CA, USA, 2010. USENIX Association.

[32] Subho S. Banerjee, Zbigniew T. Kalbarczyk, and Ravishankar K. Iyer. Sequences to systems: Accelerated genomic analytics at scale. Technical report, Coordinated Science Laboratory Technical Report, 2016.

[33] Broad Institute. The gamgee library for genomics data processing and analysis. http://gatkforums.broadinstitute.org/wdl/discussion/4767/conference-talk-cppcon-2014-sep-8-the-gamgee-library-for-genomics-data-processing-and-analysis. Accessed: 2016-02-16.

[34] Ben Langmead, Michael C Schatz, Jimmy Lin, Mihai Pop, and Steven L Salzberg. Searching for SNPs with cloud computing. *Genome Biology*, 10(11):R134, 2009.

[35] Michael C Schatz. CloudBurst: Highly sensitive read mapping with MapReduce. *Bioinformatics*, 25(11):1363–1369, 2009.

[36] Markus Hsi-Yang Fritz, Rasko Leinonen, Guy Cochrane, and Ewan Birney. Efficient storage of high throughput DNA sequencing data using reference-based compression. *Genome Research*, 21(5):734–740, 2011.

[37] Andrew Lamb, Matt Fuller, Ramakrishna Varadarajan, Nga Tran, Ben Vandiver, Lyric Doshi, and Chuck Bear. The Vertica analytic database: C-store 7 years later. *Proceedings of the VLDB Endowment*, 5(12):1790–1801, 2012.

[38] Arjun P. Athreya, Subho S. Banerjee, C. Victor Jongeneel, Zbigniew T. Kalbarczyk, and Ravishankar K. Iyer. Decomposing genomics algorithms: Core computations for accelerating genomics analyses. Technical Report UILU-ENG-14-2201, Coordinated Science Laboratory Technical Report, 2014.

[39] N. G. de Bruijn. A Combinatorial Problem. *Koninklijke Nederlandsche Akademie Van Wetenschappen*, 49(6):758–764, June 1946.

[40] Richard Durbin, Sean R Eddy, Anders Krogh, and Graeme Mitchison. *Biological Sequence Analysis: Probabilistic Models of Proteins and Nucleic Acids*. Cambridge Univ Press, 1998.

[41] Guy Cochrane, Charles E Cook, and Ewan Birney. The future of DNA sequence archiving. *GigaScience*, 1(1):2, 2012.