FPGA Accelerated INDEL Realignment in the Cloud

Lisa Wu[*], David Bruns-Smith[*], Frank A. Nothaft[†], Qijing Huang[*], Sagar Karandikar[*], Johnny Le[*], Andrew Lin[*],
Howard Mao[*], Brendan Sweeney[*], Krste Asanović[‡], David A. Patterson[§], and Anthony D. Joseph[*]
[*]*Computer Science Division, EECS Dapartment, University of California, Berkeley, CA*
{*lisakwu, bruns-smith, fnothaft, qijing.huang, skarandikar, johnny.le, andrewhlin, zhemao, krste, pattrsn, adj*}@*berkeley.edu*
[†]*Databricks* [‡]*UC Berkeley and SiFive, Inc.* [§]*UC Berkeley and Google, Inc.*

***Abstract*—The amount of data being generated in genomics is predicted to be between 2 and 40 exabytes per year for the next decade, making genomic analysis the new frontier and the new challenge for precision medicine. This paper explores targeted deployment of hardware accelerators in the cloud to improve the runtime and throughput of immense-scale genomic data analyses. In particular, INDEL (INsertion/DELetion) realignment is a critical operation that enables diagnostic testings of cancer through error correction prior to variant calling. It is the slowest part of the somatic (cancer) genomic analysis pipeline, the alignment refinement pipeline, and represents roughly one-third of the execution time of time-sensitive diagnostics for acute cancer patients.**

To accelerate genomic analysis, this paper describes a hardware accelerator for INDEL realignment (IR), and a hardware-software framework leveraging FPGAs-as-a-service in the cloud. We chose to implement genomics analytics on FPGAs because genomic algorithms are still rapidly evolving (e.g. the de facto standard "GATK Best Practices" has had five releases since January of this year). We chose to deploy genomics accelerators in the cloud to reduce capital expenditure and to provide a more quantitative performance and cost analysis. We *built* and *deployed* a sea of IR accelerators using our hardware-software accelerator development framework on AWS EC2 F1 instances. We show that our IR accelerator system performed 81× better than multi-threaded genomic analysis software while being 32× more cost efficient.

***Keywords*-Computer Architecture, Microarchitecture, Accelerator Architecture, Hardware Specialization, Genomic Analytics, INDEL Realignment, FPGA Acceleration, FPGAs-as-a-service, Cloud FPGAs**

I. Introduction

Precision medicine is at the forefront of scientific and empirical diagnostic testing for time-sensitive diagnoses and treatments such as acute cancer patients, neonatal intensive care unit patients, and infectious disease patients. Taking acute cancer patients as an example, there are more than 20,000 new cases of Acute Myeloid Leukemia (AML) every year in the United States, and the overall five-year survival rate is only 26% [1]. AML stratifies into several different mutational subtypes, and optimal treatment strategies depend on the mutations that a given patient has. In order to correctly match a patient with their optimal treatment, somatic variant calls (i.e. identified cancer mutations) must contain as few errors as possible [2], [3]. In AML, time is critical, as a patient presenting in acute blast crisis can die within days, so a few hours difference in obtaining the genomic analysis results can affect the timely treatment of the patients.

Fifteen years ago, using genome sequencing as a diagnostic test would cost around $100 million, an unaffordable luxury for most patients. The advancement of NGS (Next-Generation Sequencing) has democratized genome sequencing and allowed the sequencing cost to drop by 100,000-fold, making it possible to sequence a human genome for less than $1000 today. This in turn has led to the generation of petabyte- and exabyte-scale genome datasets in both industrial and academic settings. In fact, the Washington Post in 2015 claimed that the biggest data challenge in the next decade will be human genomics data, predicting that as many as 1 billion people will have their genomes sequenced by 2025, generating up to 40 exabytes a year of genomic data [4]. To keep up with this data deluge, prior work has focused on providing more efficient software frameworks [5], [6], or more efficient hardware using specialization [7], [8], [9], [10], [11], [12], to speed up the *primary alignment* analysis pipeline, where the DNA reads are aligned to a reference sequence of the species. In particular, the compute-intensive Smith-Waterman seed extension dynamic programming algorithm and the suffix array lookup algorithm are well understood by computer architects, and have been accelerated via FPGA and ASIC implementations.

Figure 1 depicts a typical genomic analysis flow, GATK3[1] [16], developed at the Broad Institute of MIT and Harvard [17] and the three major genomic analysis pipelines. Figure 2 shows the execution breakdown of each analysis pipeline. In order to detect genomic edits and variants (*variant calling*), such as genetic mutations that are associated with cancer, at a reasonable accuracy, the reads and alignments produced after primary alignment need to be processed through *alignment refinement* to correct systematic and correlated errors present in the primary sequencing data and their alignments. These error correction steps are necessary as these reads and alignments con-

[1] Newly released GATK4 uses a different pipeline that does not use INDEL realignment, but is only suitable for germline (non-cancer) variant calling [14], [15].

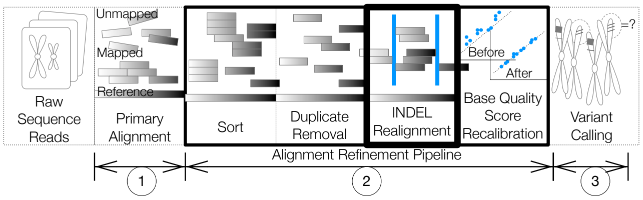


Figure 1: A typical genomic analysis flow contains three pipelines: ① *primary alignment* or read mapping, ② *alignment refinement* or read preprocessing, and ③ *variant calling*. Here, the DNA from a biological sample is first sequenced and then aligned to a known reference genome. We then apply several alignment refinement steps (lightly-bolded) to correct errors and biases in the reads, before identifying the sequence variants [13]. Our acceleration target is the INDEL realignment (heavily-bolded), the most expensive step in the alignment refinement pipeline.

tain 0.5%–2% errors, while the variants exist only at a frequency of one-in-one-hundred-thousand base pairs, and cannot be accurately detected pre-refinement. Not only is the alignment refinement analysis pipeline important, it runs approximately four times slower than the primary alignment pipeline (see Section II for detailed execution time breakdown). In addition, the refinement algorithms are difficult to understand, resulting in a lack of hardware accelerators proposed for this pipeline to date.

For applications that have broad applicability, stable algorithms, and a large customer base, it makes sense to design and fabricate ASICs to capitalize on the exceptional performance and energy efficiency such architectures can offer. A poster child of such architecture is Google's TPU for machine learning workloads deployed in 2015 [18]. However, not all application domains have large enough usage, or algorithms stable enough, to amortize the expensive ASIC manufacturing and deployment cost. For these applications, leveraging FPGAs in the cloud seems a natural fit that not only allows rapid prototyping and deployment of new architectures, but also offers these accelerators transparently to the users and customers who do not know how to program an FPGA. A successful example of this class of applications is the Bing search engine deployed in Microsoft Catapult [19].

In this work, we propose genomic analytics as another example that leverages the synergy between domain-specific architectures and FPGAs-as-a-service in the cloud. We make the following contributions:

- A throughput-efficient hardware accelerator for INDEL realignment, called the IR accelerator. An IR accelerator incorporates specialized datapaths and a specialized local memory subsystem. With our implementation, a sea of 32 IR accelerators fit on a Xilinx Virtex UltraScale+ FPGA [20] and can process up to 4 billion base pair comparisons per second, with INDEL realignment performance measured at > 80 times faster than software running 8 threads. To our knowledge, this is the first work to detail the internals of an FPGA accelerated genomics analytics system that does not target the primary alignment pipeline.
- An in-depth tour of the FPGA accelerated IR *system* design process, hardware and software components that allow the deployment of an end-to-end system evaluation, and various microarchitecture and system optimizations that allow an order of magnitude performance efficiency over even the most optimized software at a fraction of the cost. Running IR for chromosomes 1–22 on our accelerated system takes a little more than 31 minutes and costs less than $1 on an AWS EC2 F1 instance [21] as opposed to more than 42 hours on GATK3 [16] for $28.

II. Background and Motivation

To begin we provide some background on INDEL realignment: its role and its importance in genomic analysis, and its software algorithm and characteristics.

A. INDEL Realignment Background

The goal of running the genomic analysis pipelines is to *call the variants*, or identify nucleotide differences, between an individual genome and the reference genome at a given position, with acceptable accuracy. These variants can then be used to identify genetic associations with diseases, mutations in cancer, or assist in population studies. After running the primary alignment pipeline, the aligned reads frequently include errors that can lead to incorrect variant calls. To eliminate these errors, we rely on several pre-processing stages, also known as the alignment refinement pipeline, that are run between the primary alignement and the variant calling pipelines.

INDEL (INsertion/DELetion) realignment, also called *local realignment*, addresses an issue inherent to the primary read mapping. In particular, if a read contains an insertion/deletion, the mapping will commonly identify the correct genomic region that a read should map to, but will locally misalign the read relative to other reads that contain the same underlying sequence variant [16]. During local realignment, we identify the most common sequence variant in the set of reads that contain insertions/deletions, and rewrite the local read alignments to ensure that all reads that contain a single sequence variant are aligned with a consistent representation. This step is necessary because the algorithms used to compute the pairwise alignment of two sequences are fundamentally probabilistic [22], [23], [24], which can lead to inconsistent representations for equivalent sequence edits [25].

There are more than fifty state-of-the-art somatic variant callers used in different standard and non-standard analysis pipelines [26]. INDEL realignment (IR) is especially important for cases where there are low-frequency somatic variants (difficult to detect) or when visualization

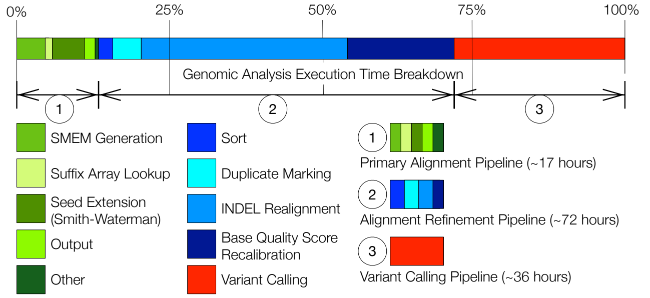


Figure 2: Portions of the primary alignment pipeline, such as the Smith-Waterman algorithm, have been the targets of hardware acceleration for quite some time because they are compute-intensive, and well understood by computer architects. However, the alignment refinement pipeline takes roughly 60% of the total execution time, is the slowest pipeline, and has not been accelerated by hardware to date. In this figure, the primary alignment execution time is measured running BWA-MEM with execution breakdown obtained from [10]. The alignment refinement and variant calling execution times are measured running GATK3[2].

and manual inspection of particular cell (re)alignments is desired (most somatic biochemists prefer manual inspection of cancer cell (re)alignments). Even though variant calling algorithms have been evolving and recently more and more algorithms have moved from position-based (e.g. IR in GATK3, Mutect1 [27]) to graph-based (e.g. HaplotypeCaller in GATK4 [15], Mutect2) and DNN-based algorithms (e.g. applied somatic DeepVariant [28]), most non-position-based algorithms are still being improved. De Brujin graph-based HaplotypeCaller in its current state produces low quality variants and cannot be used for somatic calling. Both Mutect1 and Mutect2 are developed as part of the GATK somatic variant callers. However, Mutect1 remains the standard as Mutect2 runs 30–50× slower. Examples of other variant callers that use IR as part of the pre-processing (refinement) steps to provide more accurate insertions and deletions are Strelka [29] and VarDict [30].

We examine the compute requirements of IR by obtaining the execution profile of GATK3's genomic analysis pipelines. For this work, we use a complete genome sequenced and well-characterized by the 1000 Genomes Project [31], [32], cancer patient NA12878 [33], mapped against the recent release of the human genome GRCh37 [34]. We measured the wall-clock time running the three genomic analysis pipelines using the open-source software analysis tools developed at the Broad Institute – BWA-MEM [35] for primary alignment, and GATK3 for alignment refinement and variant calling[2]. As shown in Figure 2, the primary alignment accounts for *less than 15%* of the genomic analysis execution time, while the alignment refinement pipeline accounts for roughly 60% of the execution time. Note that the execution profile will be different for other standard or non-standard variant calling pipeline. This execution profile is served to show how time consuming IR can be, and therefore pipelines that use IR or similar calculations to IR can benefit from such hardware acceleration. Some variant callers previously infeasible to run can now be used with much less investment in time and money once IR portion of the pipeline is accelerated.

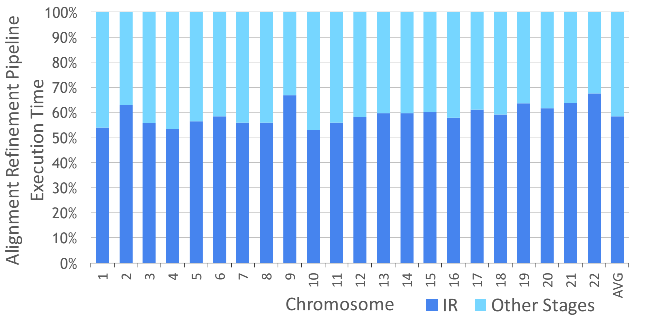


Figure 3: INDEL realignment is a key operation that is compute-intensive and a natural fit for acceleration. Here we see IR consuming on average 58% of the alignment refinement pipeline execution time on GATK3. With the most popular genomic analysis pipelines spending roughly three-fifth of the execution time on alignment refinement, we estimate that IR accounts for roughly one third of genomic analysis execution time.

We further measured the fraction of alignment refinement execution time attributed to IR using GATK3 as shown in Figure 3. Ranging from 53% to 67%, alignment refinement spends an average of 58% of its execution time in INDEL realignments. Given the measurement shown in Figure 2, IR accounts for roughly 34% of the total genomic analysis execution time, a significant portion of the total runtime that if accelerated, can provide remarkably better speedup than accelerating portions of the primary alignment pipeline such as the Smith-Waterman seed extension (5%) or suffix array lookup (1.5%).

B. *INDEL Realignment Algorithm*

We assume the readers have enough background on genomics-specific terms to understand the INDEL realignment algorithms. Please see a pictorial representation and glossary in the Appendix if that is not the case.

Algorithm The pseudo code to compute one INDEL realignment target is presented in Algorithms 1 and 2. First, it computes minimum weighted Hamming distances (*min_whds*) for each read against each consensus (including the reference) for a given target. Second, Algorithm 2 performs the "scoring" of each consensus, selects the best consensus, and realigns the reads if necessary. Since Hamming distance measures how different two strings are, the consensus with the smallest Hamming distances against all

[2] Data collected on an AWS r3.2xlarge EC2 instance: a quad-core Ivy Bridge server (Intel Xeon E5-2670 v2, 4C/8T, 2.5GHz) with 61GiB memory and 160GB SSD. We chose this type of EC2 instance because GATK3 does not scale beyond 8 threads.

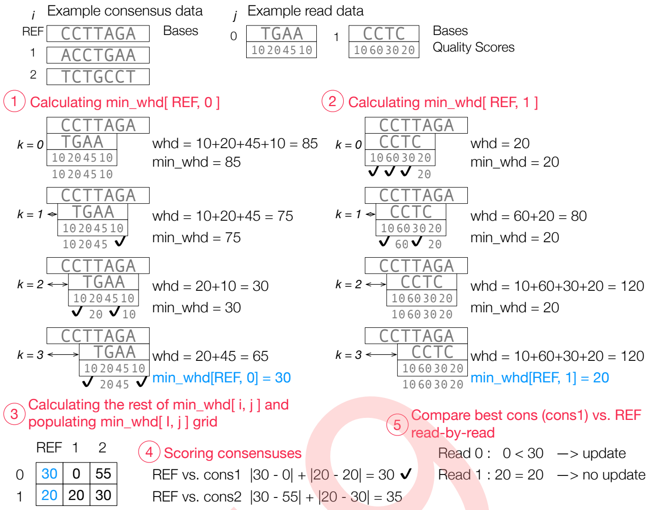
Algorithm 1: Minimum Weighted Hamming Distances

```
input  : An array of consensuses $cons$ of size $NumConsensuses$
input  : An array of reads $reads$ of size $NumReads$
output : An array of minimum weighted hamming distance $min\_whd$ for
         every pair of $(cons[i], reads[j])$ and its associated index offset
         $min\_whd\_idx$
// Part 1: Minimum Weighted Hamming Distances
// This algorithm calculates the minimum weighted
// distances for all reads against all consensuses
// including the reference (i=0)
function Min_WHD
for $i = 0..(NumConsensuses - 1)$ do
    for $j = 0..(NumReads - 1)$ do
        // do a sliding comparison of the read along
           the consensus
        for $k = 0..(cons[i].length - reads[j].length - 1)$ do
            $curr\_whd \leftarrow$ Calc_WHD ($cons[i], reads[j], k$) ;
            // update the minimum whd
            // if the current whd is smaller
            if $curr\_whd < min\_whd[i, j]$ then
                $min\_whd[i, j] \leftarrow curr\_whd$;
                $min\_whd\_idx[i, j] \leftarrow k$;

input  : A consensus sequence (bases) $cons[i]$
input  : A read sequence (bases) $reads[j]$ and its corresponding quality
         scores $quals[j]$
input  : A starting index $k$
output : A weighted hamming distance $whd$
// Part 1.1: Calculate Weighted Hamming Distance
// This algorithm compares read bases against
// consensus bases starting at index $k$ and sums the
// corresponding quality scores if the bases don't
   match
function Calc_WHD
for $n = 0..(reads[j].length - 1)$ do
    $m \leftarrow n + k$;
    if $cons[i][m] \neq reads[j][n]$ then $whd \leftarrow whd + quals[j][n]$;
```

Algorithm 2: Consensus Selection and Read Realignment

```
input  : An array of minimum weighted hamming distance $min\_whd$ for
         every pair of $(cons[i], reads[j])$
output : A picked consensus $best\_cons$
output : An array of booleans $realign[j]$, TRUE/FALSE per read, to
         indicate if an update/realignment needs to be performed
output : An array of new read positions $new\_pos[j]$, one position per
         read, to be updated if realignment is needed
// Part 2: Consensus Selection and Read Realignment
// This algorithm scores each consensus against
// the reference cons[0] by summing the absolute
// differences on the whds for all reads
function Score_n_Select
for $i = 1..(NumConsensuses - 1)$ do
    for $j = 0..(NumReads - 1)$ do
        $absdiff = |min\_whd[i, j] - min\_whd[0, j]|$;
        $scores[i] \leftarrow score[i] + absdiff$;
    // update the best consensus
    // if the current consensus score is smaller
    if $score[i] < score[best\_cons]$ then $best\_cons \leftarrow i$ ;
// update the read alignments if the picked
// consensus scores better than the reference
function Reads_Realignments
for $j = 0..(NumReads - 1)$ do
    $realign[j] \leftarrow$ FALSE;
    if $min\_whd[best\_cons, j] < min\_whd[0, j]$ then
        $realign[j] \leftarrow$ TRUE;
        $new\_idx \leftarrow min\_whd\_idx[best\_cons, j]$;
        $new\_pos[j] \leftarrow new\_idx + target\_start\_pos$;
```



i Example consensus data
REF CCTTAGA Bases
1 ACCTGAA
2 TCTGCCT
j Example read data
0 TGAA 10 20 45 10
1 CCTC 10 60 30 20
Bases
Quality Scores
1 Calculating min_whd[REF, 0]
k = 0 whd = 10+20+45+10 = 85 min_whd = 85
k = 1 whd = 10+20+45 = 75 min_whd = 75
k = 2 whd = 20+10 = 30 min_whd = 30
k = 3 whd = 20+45 = 65 min_whd[REF, 0] = 30
2 Calculating min_whd[REF, 1]
k = 0 whd = 20 min_whd = 20
k = 1 whd = 60+20 = 80 min_whd = 20
k = 2 whd = 10+60+30+20 = 120 min_whd = 20
k = 3 whd = 10+60+30+20 = 120 min_whd[REF, 1] = 20
3 Calculating the rest of min_whd[i, j] and populating min_whd[I, j] grid
REF 1 2
0 30 0 55
1 20 20 30
4 Scoring consensuses
REF vs. cons1 |30 - 0| + |20 - 20| = 30 ✓
REF vs. cons2 |30 - 55| + |20 - 30| = 35
5 Compare best cons (cons1) vs. REF read-by-read
Read 0 : 0 < 30 —> update
Read 1 : 20 = 20 —> no update

Figure 4: An INDEL realignment example with 3 consensuses and 2 reads. Consensus 1 was picked as the best consensus, and only Read 0 was updated because the best consensus's Read 1 did not have a better (i.e. smaller) *min_whd* than the REF.

the reads is the consensus that exhibits the most similarities with all the reads, and therefore is the "best" or the "picked" consensus. The last part of the algorithm compares the best consensus with the reference read-by-read. When the picked consensus exhibits more similarity to the read than the reference, i.e. has a smaller Hamming distance, the read is updated with the realigned attributes, such as its read start position and mapping quality score.

INDEL Realignment Example We walk through a simple example of INDEL realignment at a locus with 3 consensuses and 2 reads as depicted in Figure 4. A typical locus can contain 2–32 consensuses and 10–256 reads. We calculate the weighted Hamming distance between two strings, using index *i* to indicate the current consensus, index *j* to indicate the current read *j*, and *k* to denote the position offset from the consensus where the starting position of the read is aligned to. For every *(i, j, k)* triple, the weighted Hamming distances are calculated as described in Algorithm 1 and illustrated in Figure 4. The consensus and read sequences are compared base-by-base: when they match, the weight is 0 and when they do not, the weight is 1. The weighted Hamming distance (*whd*) is then simply calculated as the sum of the quality scores when the bases do not match. Once the *whds* for all of the aligned position combinations between consensus *i* and read *j* are calculated, i.e. *k* is shifted from 0 to *(length of consensus i - length of read j + 1)*, the minimum weighted Hamming distance over all *k* is recorded into a two-dimensional *(i, j)* grid along with the offset *k*. When the grid is completed, a score is calculated for each consensus against the reference by summing the absolute differences between *min_whd[REF, j]* and *min_whd[i, j]* for all reads. The lowest scoring consensus is then picked as the best consensus. As the final step, we update the read start positions for all reads where the consensus scored better.

C. Software Performance

We analyze the software performance for the IR algorithms and show that the execution of INDEL realignment is compute-bound. Specifically, it is bottlenecked on the weighted Hamming distance calculation in Algorithm 1. To be consistent, we use the same notation to represent the ith consensus as $cons[i]$, and the jth reads as $reads[j]$. We use C to denote the number of consensuses in a target, and R to denote the number of reads in a target. For every $(cons[i], reads[j])$ pair, $whds$ are calculated over all possible alignments between the consensus and the read. Having a consensus length of m and a read length of n, there are $m - n + 1$ possible alignments, and therefore a worst case of $m - n + 1$ possible $whds$ to compute and from which to pick the min_whd.

Thus Algorithm 1 has a worst case complexity of $O(CR \cdot (m - n + 1) \cdot n)$ where $C \leq 32$, $R \leq 256$, $m \leq 2048$, and $n \leq 256$; an astonishing worst case of $3,684,352,000$ comparisons for just calculating the $whds$ for *one* IR target. To put the number of computations into perspective, for even the smallest human chromosome (`Ch21`), there are more than $48,000$ IR targets of varying sizes. Assuming we can perform one read base comparison and one quality score accumulation per cycle, and assuming that we can adequately feed the `Calc_WHD` function with at least 3 bytes per cycle (1 byte each for consensus base, read base, and read quality), this portion of the algorithm becomes completely compute-bound.

In the next section, we show how to overcome this compute-bound, big data challenge by employing three microarchitectural design principles: *data reuse* by utilizing local storage, *task parallelism* by instantiating multiple lightweight IR units, and *computation pruning* by eliminating unnecessary calculations whenever possible.

A question that arises when we accelerate compute-intensive workloads is why not use CPU SIMD units or GPUs. For the genomic analysis INDEL realignment kernels we studied, SIMD units have limited applicability. We can try mapping SIMD units when we are calculating the weighted Hamming distances between a read and a consensus sequence. However, this process does not map efficiently to SIMD because the sequences will only be aligned in an n-word wide SIMD unit at 1-out-of-n indices in a loop, which will likely lead to poor SIMD performance. Further, the genome sequenced reads follow a Zipf-like distribution at roughly between 100 reads and 100,000 reads per location interval. This extremely imbalanced data distribution will likely trigger significant thread divergence when run on a GPU, resulting in poor performance.

INDEL Realignment Accelerator Instructions
`ir_set_addr <buffer index> <mem addr>` Set the starting read from/write to memory address at `mem addr` for buffer number `buffer index`.
`ir_set_target <target addr>` Set the starting read position of the current target to `target addr`.
`ir_set_size <# consensuses> <# reads>` Set the number of the consensuses and the number of reads to `# consensuses` and `# reads` respectively.
`ir_set_len <consensus id> <consensus length>` Set the length of consensus `consensus id` to `consensus length` bytes.
`ir_start <unit id>` Start the INDEL realigner unit specified by `unit id` using currently configured settings.

RoCC Instruction Format

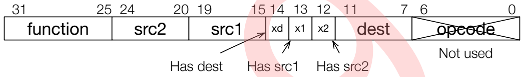


Table I: Using a simple instruction format such as the one proposed by the RoCC custom instruction format allows us to manage the INDEL realigner using just five instructions. The *opcode* field is used to encode different accelerator types. Since the accelerated IR system only contains the IR accelerator, the *opcode* field is essentially not used. The *function* field is used to encode different accelerator configurations for a given accelerator type.

III. Architecture and Microarchitecture

Here we describe the architecture and microarchitecture of a system that incorporates a sea of INDEL realignment accelerators.

A. INDEL Realignment Accelerator

The INDEL realigner is microarchitected with the design goal to maximize throughput. As each IR target can be processed completely independently without communication with other targets, we also opted to design smaller units to leverage target (i.e. task) parallelism. For the dataset we used, there is ample target parallelism: the smallest chromosome (`Ch21`) has over 48,000 targets while the largest chromosome (`Ch2`) has over 320,000 targets. We made design choices that allow us to fit as many units on an FPGA as possible while meeting the timing constraint of 125MHz, one of the clock recipes offered by the F1 instances [36]. For the FPGA on the F1 instances, the Xilinx Virtex UltraScale+, we were able to instantiate up to 32 IR units with 1 DDR channel, achieving a block RAM [37] utilization close to 90%[3].

Instruction Set Architecture The IR accelerator is managed via the five commands shown in Table I. `ir_set_addr` is invoked five times per target to set three input buffer addresses (consensus base, read base, and read

[3] For the optimized IR units, we were able to push the block RAM utilization to 87.62%, instantiating 32 units. For that design point, the CLB logic utilization is 32.53%.

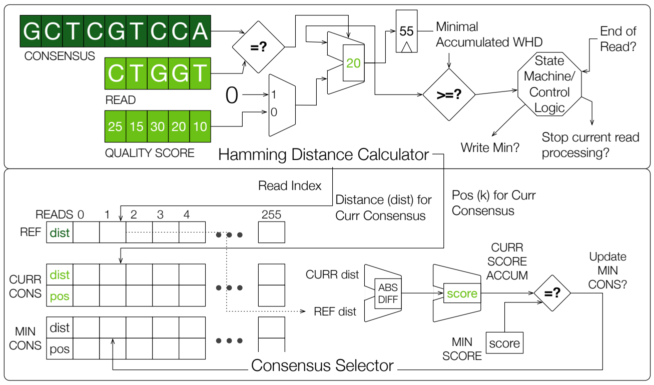


Figure 5: The INDEL realigner unit separates the IR computation into two stages. The *Hamming distance calculator* computes the minimum weighted Hamming distance for each consensus-read pair. The *consensus selector* computes a score for each consensus by summing the absolute differences in the weighted distances across all the reads between the current consensus and the reference consensus. It then finds the consensus with the minimum score (i.e. the best score) and computes a new position for each read using the realignment offsets from that consensus.

quality score) and two output buffer addresses (realignment and new read positions). `ir_set_target` is invoked once per target to set the target starting position for calculating the final read update positions. `ir_set_size` tells the accelerator how many consensuses and how many reads are in this target so the data can be transferred to the FPGA memory and then to the block RAM inside the IR unit accordingly. `ir_set_len` is invoked as many as 32 times per target, depending on how many consensuses there are. This configuration allows the IR unit to stop computing the weighted Hamming distances when the last position of the read reaches and last position of the consensus. Once the configurations are programmed based on the specific target, `ir_start` is invoked to start IR's realignment calculation.

We chose to leverage the open RoCC (Rocketchip Custom Coprocessor) [38] instruction format because it is a fixed-length format that is simple to decode. We are also able to leverage the *RoCC command router* (more details in Section III-B) coded in Chisel [39] from the open-source Rocket Chip Generator [40] to communicate from the host to the IR accelerators.

Microarchitecture Each INDEL realignment accelerator unit has two stages as depicted in Figure 5. The first stage is a `Hamming Distance Calculator` that has three input buffers, each with a read pointer, for the current consensus, read, and the associated quality score of the read being evaluated. The calculator has a simple comparator to process one base pair per cycle and perform a quality score accumulate when the base pair mismatches. A state machine outputs the minimum weighted Hamming distance for the consensus-read pair under evaluation when a read ends, along with the offset at which the minimum was computed, and the read index for the next stage.

The second stage is a `Consensus Selector`. It is performed less frequently than the calculator as the selector only needs to run once per consensus-read pair. The selector has 3 read-length buffers (256 in our implementation) to store minimum weighted Hamming distances (`dist`) and offsets (`pos`) needed to compute the score. The distances and offsets are kept for the reference, current consensus, and the running minimum consensus as shown in Figure 5-bottom. Scores are calculated by summing the absolute differences of the *min_whds* across all reads between the reference and the consensus being scored. Because the selector constitutes a small percentage of the runtime, the buffers only support one read or one write per cycle (one read/write port).

Data Reuse As discussed in Section II-B, for a specific IR site, we need to calculate $m + n - 1$ weighted Hamming distances (*whds*) for each consensus-read pair, where m is the length of the consensus and n is the length of the read. Hence, calculating all $whds$ for consensus-read pairs across C number of consensuses and R number of reads would require reusing every consensus $R \cdot (m - n + 1)$ times in the calculation, and every read $C \cdot (m - n + 1)$ times. We architected three input buffers, a consensus buffer that contains up to 32 consensuses of max length 2048 bytes each, a read base buffer and a read quality score buffer that contains up to 256 reads of max length 256 bytes each to employ data reuse in our design. We implemented these local buffers using the FPGA block RAMs. These provide fast access to reused data and eliminate the need to read/write data frequently from and to the slower FPGA-attached DRAM. When the local buffers can be accessed every cycle, as is the case in our design, it allows maximum throughput on the compute logic without having to design complicated control logic.

Although the bases can be implemented using 3 bits to represent `A,C,T,G,N`, we chose to use 1 byte for each consensus base, each read base, and each quality score in our design. This enables byte- and block-aligned reads from memory and simple data manipulation such as index decoding and masking. The input buffers for the consensuses and the reads are block-indexed and byte-selected to avoid having to shift large, random amounts when the weighted Hamming distance calculation is finished for a particular read with the consensus at the last offset $k = m + n - 1$, and starting the next read with the consensus back at the first offset $k = 0$.

Task Parallelism Since the INDEL realignment calculation is completely independent from one locus to the next, we are able to take advantage of this target parallelism and design small and throughput-oriented IR units that can be

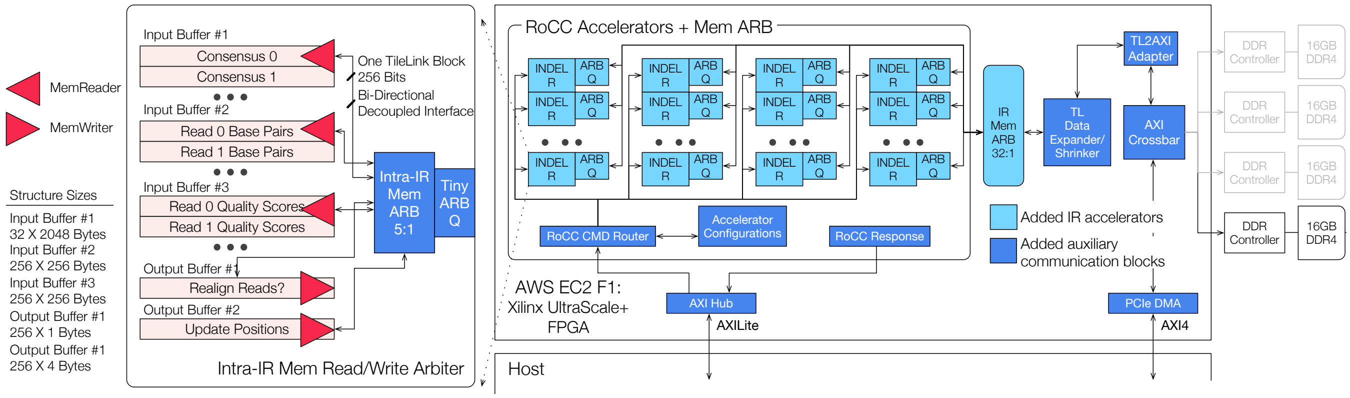


Figure 6: This figure shows the organization of a sea of 32 IR accelerators (lighter blocks) and the auxiliary communication and memory subsystem (darker blocks) on the FPGA. To communicate to and from the FPGA-attached memory, each accelerator has five memory channels as shown in the *Intra-IR Mem* diagram on the left; these channels are arbitrated into a single memory channel. The per-IR unit memory channel, connected via the *ARB Q*'s shown in the diagram on the right, are then coalesced by the *IR Mem ARB 32:1* before getting to the *AXI crossbar* and the *DDR controllers*. To communicate to and from the host, the accelerator system uses a 32-bit AXILite interface in conjunction with a *RoCC command router* as well as *accelerator configurations*. Having a separate 512-bit DMA communication between the host and the FPGA memory is key to hiding the bulk of the data transfer latency.

replicated as many times as they will fit on an FPGA. The opportunity and drawback of this approach is the scheduling of the IR targets to run on the available IR units. The computations required per target vary significantly and can lead to performance degradation if not scheduled properly, i.e. having all units wait for the slowest unit to finish before accepting new targets (see Section IV for scheduling optimizations).

Computation Pruning For a consensus length of m base pairs and a read length of n base pairs, we need to perform a worst case of n weighted Hamming distance calculations for $m - n + 1$ different offsets. However, at the end of the Hamming distance calculation, the consensus selector only needs to know the minimum accumulated Hamming distance for a consensus-read pair, and the offset where the minimum occurred. This presents an opportunity to store the running minimum Hamming distance, and stop computing the rest of the distances when it exceeds the current minimum; we call this *computation pruning*. Computation pruning eliminates $>50\%$ of the computations from the input data set we used, making the number of distance calculations and the number of offsets that need computation a lot smaller. This provides a large performance gain with minimal amount of hardware – a small register to keep track of the running minimum Hamming distance and some relatively trivial control logic to compare and stop the computation when the running sum exceeds the recorded minimum.

B. System Architecture and Integration

Figure 6 shows a block diagram of the major components in a system with INDEL realignment acceleration deployed on an AWS EC2 (Amazon Web Services Elastic Compute Cloud) F1 instance (f1.2xlarge) [21]. Each instance consists of a commodity server blade with a Xilinx Virtex UltraScale+ FPGA [20] and 4 channels of DDR4 totaling 64GB of FPGA memory. The FPGA communicates with both the host processor and the FPGA-attached DRAM via AXI4 interfaces, an open standard, on-chip interconnection specification introduced by ARM [41] for functional blocks in an SoC (System on Chip).

In Figure 6-right, shaded blocks in the diagram indicate the units designed and implemented for this work with lighter blocks presenting accelerator units and darker blocks presenting auxiliary communication units. It specifically shows the configuration with 32 IR units, and how they communicate with memory. Figure 6-left shows a zoomed-in view of how one IR accelerator is connected to memory. Each IR accelerator unit has five memory channels – three *MemReaders* connected to three input buffers (*consensus*, *read bases*, and *read quality scores*) and two *MemWriters* connected to two output buffers (*realignment* and *new positions*) – decoupling the accelerator core from the rest of the system.

The processor in the host machine copies data over DMA for PCIe [42], [43] and a 512-bit AXI4 interface [44]. It coordinates execution by issuing commands to the accelerators over a 32-bit AXI4Lite interface. The accelerators are managed using custom RoCC commands and simple state machines. The *AXI hub* converts RoCC commands and responses to and from AXILite using Memory-Mapped IO (MMIO) registers. The MMIO registers implement a ready/-valid interface and queues for commands and responses so that the host can asynchronously add a new command to the queue, or poll when awaiting a response. The *RoCC command router* routes the command to the corresponding IR Unit.

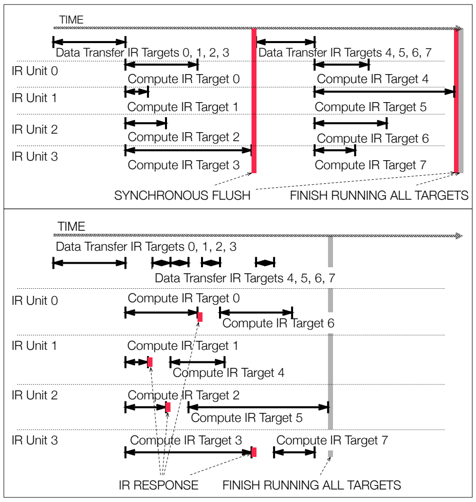


Figure 7: Instead of flushing all IR units at once in a synchronous-parallel (top) scheme, we employ the higher-performing asynchronous-parallel (bottom) scheme to launch a new target as soon as a unit becomes free.

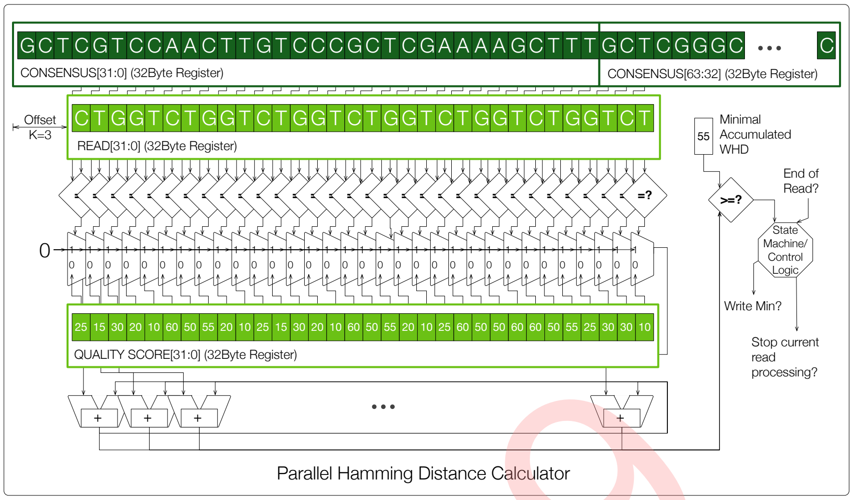

GCTCGTCCAACTTGTCCCGCTCGAAAAGCTTTGCTCGGGC ••• C
CONSENSUS[31:0] (32Byte Register)
CONSENSUS[63:32] (32Byte Register)
Offset K=3
CTGGTCTGGTCTGGTCTGGTCTGGTCTGGTCT
READ[31:0] (32Byte Register)
55
Minimal Accumulated WHD
=?
>=?
End of Read?
State Machine/ Control Logic
0
Write Min?
Stop current read processing?
25 15 30 20 10 60 50 55 20 10 25 15 30 20 10 60 50 55 20 10 25 60 50 50 60 60 50 55 25 30 30 10
QUALITY SCORE[31:0] (32Byte Register)
+
•••
Parallel Hamming Distance Calculator

Figure 8: Parallel Hamming distance calculator implements up to 32 base byte-compares and 32 quality score byte-accumulates per cycle. It stores 2 32-byte consensus blocks and 1 32-byte read block in registers to allow for 1 32-byte shift and compare every cycle. The logic for an offset of $k = 3$ is shown in this diagram.

On the host, the genomic data inputs are organized in consecutive `malloc`'ed memory arrays of one byte per base or per quality score for the three inputs: consensuses, reads, and quality scores, for streaming processing. The memory arrays for a particular target are then transferred over to the FPGA-attached DDR before starting the accelerator unit. We only instantiate one out of the four available DDR channels on F1 because even the largest target does not occupy more than 16GB of memory. This allows us to trade memory controller area and wiring for more IR compute units.

In designing the communications between the host, FPGA memory, and the IR units, we leveraged existing infrastructure available from the open-source Rocket Chip Generator [40]. Rocket Chip provides a library of parametrized hardware modules including the *RoCC command router* described above, an implementation of the AXI interface with supporting modules used to implement the *AXI Hub*, and an implementation of the TileLink interface [45] with supporting modules used to handle communication between units and the memory arbiter.

TileLink is an open source interconnect protocol. We choose to leverage the protocol for communication with the units because the existing Rocket Chip implementation is well supported and highly parametrizable, allowing us to test out different bus width without having to change the source code, while also providing access to crossbars, buffers, and width adapters [40], [39]. We used the parametrized implementation to explore a number of TileLink interface widths, and found that a 256-bit interface provided the best performance under the timing constraints.

IV. Microarchitectural Optimizations

Leveraging parallelism in an FPGA design is critical for achieving higher performance because FPGA clock speeds are an order of magnitude lower than ASICs or commodity general-purpose processors. Since each IR target input and computation is completely independent, the available parallelism trivially scales up with the volume of hardware. For a single FPGA with many IR units, the time to transfer target input, configure, and start each individual unit is negligible, so the computation time scales (almost) linearly with the number of units available. In fact, in our measurements, using PCIe DMA to transfer target input data from the host to the FPGA accounts for only 0.01% of the total runtime. The number of IR units that can be instantiated on a single FPGA is limited by the number of block RAM cells available because we leverage data reuse aggressively as described in Section III-A. We were able to fit 32 units on a Xilinx UltraScale+, pushing the block RAM utilization close to 90%.

Aside from maximizing task parallelism, we looked at three other microarchitectural optimizations to increase the performance of the accelerated IR system: scheduling targets to run *asynchronously*, increasing the *frequency* of the design, and increasing parallelism by employing *data parallelism* in addition to task parallelism.

Asynchronous Scheduling Our initial design with 32 IR units transfered data needed for all 32 targets and then launched all 32 accelerator units at once. The system waited for all units to finish before flushing the hardware synchronously and starting a new batch of targets. This synchronous-parallel scheme had minimal drawbacks because the targets could be sorted by read and consensus sizes to ensure that all the targets that are scheduled in the same batch have similar runtimes. However, the aggressive pruning logic, described above in Section III-A, causes wide variance even between targets that have the same size.

We ran a toy experiment to illustrate the effectiveness of the synchronous-parallel scheme. In this experiment, we used 8 same-sized INDEL realignment targets that contain 2 consensuses and 8 reads each (stripped down from real targets in `Ch22`). Figure 7-top is an annotated reproduction of the timing diagram generated during IR execution. The first chunk of data transfer time includes transferring data to the FPGA memory for targets 0–3 and starting IR units 0–3, each running a target in parallel. Notice that the compute time for target 3 is about 8 times longer than the compute time of target 1, resulting in 3 out of 4 units idling for a majority of the total runtime.

This variance necessitated a performance optimization that launched IR targets asynchronously as demonstrated in Figure 7-bottom. The asynchronous-parallel scheme allows a scheduled target to be launched as soon as an IR unit is free. To implement the asynchronous scheme, a response is sent from each accelerator when the computation is complete. The host can then poll the MMIO register "response valid" signal and start the next scheduled target. This optimization allows us to keep all 32 units busy most of the time, improving efficiency of the IR system.

Frequency By default, the IR accelerator system is built with a 125 MHz clock recipe [36]. However, F1 also makes a 250 MHz clock recipe available. We examined the trade-offs between the performance benefits of doubling the clock frequency and the changes necessary to the system to meet the more stringent timing requirements. At 250 MHz, the critical timing path is over 95% routing delay resulting in violated paths within the AXI4 memory system. This is due to having a memory system that must service 32 different accelerator units. Even at 125 MHz, the majority (over 90%) of the critical path consists of routing delay. Therefore, instead of doubling the clock frequency, we recognized that a significant amount of combinational logic could be added to the system at 125 MHz without having a significant impact on timing requirements.

Data Parallelism From the software performance analysis in Section II-C, the weighted Hamming distance calculation in Algorithm 1 is by far the most significant performance bottleneck, due to the sheer number of base pairs that need processing. We added combinational logic to increase the number of Hamming distance comparisons performed per-cycle inside each individual IR accelerator unit as shown in Figure 8. We leverage the fact that we can read 32 bytes of data from the block RAM per cycle to perform 32 weighted Hamming calculations (32 compares and 32 accumulates) in parallel. However, the minimum Hamming distance is calculated by shifting the read along the consensus, and for a given read chunk (i.e. 32 bytes), the corresponding consensus chunk for comparison will not necessarily be aligned to the 32-byte memory blocks. Thus we pipeline the inputs so that two 32-byte memory blocks from the consensus are available each cycle without adding an additional read port to the input buffers.

Machine Configurations

EC2 Instance	System Components	
f1.2xlarge	Host Processors	Intel Xeon E5-2686 v4 (Broadwell) 4C/8T, 2.2 GHz
	Host Memory	122 GiB
	FPGA	1x Xilinx Virtex UltraScale+ VU9P 2.5 M logic elements, 6,800 DSPs
	FPGA Memory	64 GB, 4xDDR4
r3.2xlarge	Processors	Intel Xeon E5-2670 v2 (Ivy Bridge) 4C/8T, 2.5GHz
	Memory	61 GiB

Table II: Hardware configurations for the AWS EC2 F1 instance used to deploy the accelerated IR system and the EC2 R3 instance used to run the open source GATK3 software implementations.

V. Accelerated INDEL Realignment System Evaluation

In this section, we compare the performance of our IR system against the de facto standard open-source analysis toolkit GATK3.

A. Methodology

We implemented the INDEL realignment accelerator in Chisel [39], configured and compiled the Chisel blocks into Verilog using the open-source Chisel toolchain, and then instantiated and connected various IP components provided by AWS/Xilinx such as the memory controllers and the AXI4 crossbar [46]. Combining the built and leveraged Verilog components, we then synthesized, placed, and routed the designs using the Xilinx-provided Vivado tool suite [47] into an FPGA image for the F1 instance called an AFI (Amazon FPGA Image). The AFI is then ready to be loaded and used anywhere in the world where users have access to an AWS EC2 F1 instance.

For the INDEL realignment workload to run on the FPGA, we wrote C APIs for encoding RoCC commands and for configuring the accelerators through the RoCC commands. AWS provides an FPGA management API that is easily modified to perform large chunks of data transfers via PCIe DMA. For configuring, starting, and flushing any accelerators, we dispatch RoCC commands via the AXILite interface. We wrote control programs in C/C++ that 1) `malloc` input/output arrays in the host memory, 2) transfer large data chunks from the host to the FPGA-attached DRAM and vice versa, 3) configure and start the accelerators one unit at a time; the *accelerator configurations module* and

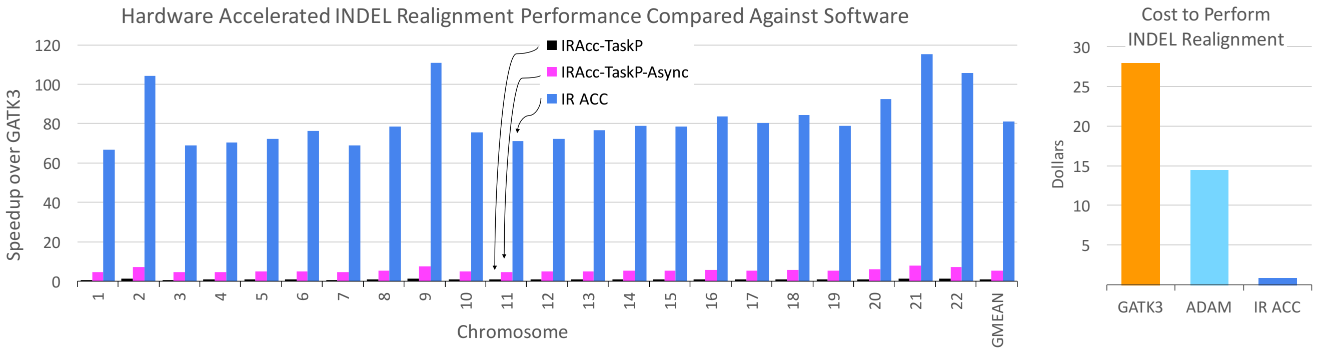


Figure 9: Our FPGA-accelerated INDEL realignment system deployed with 32 IR accelerators (`IRAcc-TaskP`), asynchronous-parallel scheme (`IRAcc-TaskP-Async`), and additional data parallelism (`IRAcc-TaskP-Async-DataP` which we will shorthand to `IR ACC`) achieved a remarkable speedup of 66.7×–115.4× over software running 8 threads (figure left). Even though it is relatively inexpensive to rent AWS EC2 R3 instances populated with old Ivy Bridge Xeon servers, our most optimized deployed system (`IR ACC`) is still 32× more cost efficient than GATK3 and 17× more cost efficient than ADAM (figure right).

the *RoCC command router* shown in Figure 6 handle this task, and 4) wait for responses and configure and start the units that are finished with the previous task. The runtime of the workload is measured *end-to-end* in the control program including data transfer and compute for all of the IR targets for each chromosome.

We run the IR workload on a well characterized NA12878 genome from the 1000 Genomes project. This genome was sequenced at 60–65× coverage producing 763,275,063 total reads, before being aligned to the HS37D5 build of the GRCH37 reference genome using BWA-MEM, the de facto standard software analysis tool for primary alignment. We then run chromosomes 1 to 22 through INDEL realignment, measure runtime in wall clock time, and show the individual chromosome results as well as the geometric mean. We compare our hardware performance against software running on a single node EC2 R3 instance. Table II lists specific machine configurations for our experiments. We chose to use the r3.2xlarge instance type with 4 cores and 8 threads per node as GATK3 cannot scale beyond 8 threads, making it the most cost efficient hardware platform available in EC2 to run the GATK3 experiments.

B. Results

Figure 9-left shows the accelerated IR system speedup over the software implementation GATK3. Number of targets vary per chromosome, and target sizes vary wildly. Averaged across all chromosomes, our system performed 81.3× better than software running 8 threads[4]. The measurement is instrumented in the control program for IR and includes input preprocessing (file I/O), data transfer to the FPGA, IR compute, and IR response. IR compute includes configuring the accelerators using RoCC commands and starting the accelerators while IR response includes flushing the accelerators and getting a response of completion out of the MMIO register.

By instantiating 32 IR units without optimizations, our accelerated IR system performed 0.7×–1.3× better than GATK3 (`IRAcc-TaskP` in Figure 9). Scheduling the IR targets asynchronously gives us an average of 6.2× performance gain (`IRAcc-TaskP-Async`), while incorporating a data-parallel version of the Hamming distance calculator (`IR ACC`) provided another 15× speedup. Even if the software can optimistically scale linearly via core and thread count, it will still take 20 32-thread instances to match our performance. This is no small feat as the FPGA is running at a relatively low clock frequency of 125MHz compared to the Xeon server at 2.5GHz on which we run our baseline software. In fact, even the recent ASIC-accelerated system, the Google TPU, was only able to achieve a 30× performance gain over software-only implementations [18].

Comparison with ADAM In addition to comparing performance with GATK3, we also run the most optimized open-source software implementation of the alignment refinement pipeline, ADAM [48], [49], for comparison[5]. Our accelerated IR system performs 30.2×–69.1× better than ADAM, with an average of 41.4× speedup over `Ch1` –`Ch22`.

Cost Comparison It is usually difficult to do an objective cost versus performance tradeoff when we evaluate computer system performance. Architects tend to use area and power to estimate the cost of running a particular system. Since Amazon has priced out AWS EC2 instances proportional to the TCO (Total Cost of Ownership) of running different types of systems, we can simply use that as the true cost

[4] The experiments are run with chromosomes interleaved and taking the average of 10 runs, mimicking real user behavior of running INDEL realignment "cold" without effects of OS caching on the host machine.

[5] We run ADAM on top of APACHE SPARK. We used the latest stable versions of all softwares – ADAM 0.22.0, APACHE SPARK 2.1.0, Scala 2.11, and Java 8.

(dollar amount) it takes to run these systems. Figure 9-right depicts the cost to run INDEL realignment on all chromosomes using GATK3, ADAM, and IR ACC. Since GATK3 and ADAM are running on 4-core 8-thread Xeon servers, they are particularly economical because these are older servers and are priced at just 66.5¢ per hour. The F1 instances are more expensive at $1.65 an hour. Our accelerated system not only performs an order of magnitude better, it is also an order of magnitude more cost efficient than running the most optimized software, and can complete INDEL realignment for all chromosomes for just 90¢. Whereas, GATK3 and ADAM take $28 and $14.5 to run on R3 instances respectively.

Comparison with HLS We implemented a version of the accelerators using the SDAccel Development flow [50], a Xilinx HLS (High Level Synthesis) flow. We built the accelerators using C IR algorithm implementation similar to the ones presented in Section II-B, and used OpenCL for communications between the host and the accelerators. However, we were only able to get a modest speedup of 1.3×–3.1× over GATK3 because of limitations on the HLS infrastructure. Xilinx OpenCL has a hard limit of 16 on the number of compute units that can be scheduled asynchronously, limiting task parallelism. HLS had difficulties extracting coarse-grained parallelism from the kernel automatically due to ambiguous memory dependencies and aliasing present in the algorithm. Once a design fails any part of the backend flow, it was difficult to decipher, debug, or optimize timing paths as the design was automatically generated from high-level abstraction with a large number of unreadable states and variables.

Comparison with GPU-based Systems Fair comparisons against GPU-based systems are difficult because there exist no GPU implementations for INDEL Realignment. For a single high-end GPU AWS EC2 instance ($3.06/hr) to match the performance and the cost of an accelerated IR system on an F1 instance ($1.65/hr), the GPU system needs to achieve a 148.36× speedup over the GATK3 baseline. Here we present a few data points for comparison. GPU-accelerated implementations performing similar calculations in the genomics domain (BarraCUDA [51] and CUSHAW2-GPU [52]) and in other domains (Viterbi decoder [53] and Iris template matching [54]) achieve 1.4–14.6× performance gains over CPU implementations. In general, GPU implementations rarely offer more than 20× speedup compared to optimized CPU implementations [55]. These comparisons are imprecise, but these GPU implementations do not come close to match the efficiency of our implementation.

VI. Related Work

Industry Implementations There is strong industrial interest in accelerating genomic analysis algorithms, as evidenced by the commercialization of hardware accelerators for genomics at both Microsoft [56] and the recently-acquired Edico [57]. Edico's DRAGEN is a closed-source industry implementation of several genome sequencing analysis pipelines on FPGAs including GATK4. They claim to provide 78-82× performance gain, matching our IR performance, but over the entirety of the analysis pipelines.

Primary Alignment and Smith-Waterman A substantial prior work focus has been on the Smith-Waterman kernel for pairwise local sequence alignment. Accelerated versions of Smith-Waterman have targeted both FPGAs [58], [59], [60] and SIMD/GPU architectures [61], [62]. Other work has targeted the whole Burrows-Wheeler Alignment (BWA) algorithm [9], [10] for FPGAs. However, Smith-Waterman accounts for only 5% of the complete genome sequencing pipeline and BWA only 15%, whereas our chosen target, INDEL realignment, accounts for a significantly longer runtime and has not been the target of previous acceleration efforts.

Other Accelerated Genome Alignment While we focus on the widely used pipeline outlined in Figure 2, other FPGA accelerators have targeted other kinds of genome alignment. The GateKeeper accelerator [8] implements an FPGA-based alignment pre-filter that reduces the amount of locations in the genome that need to be queried. Darwin is an FPGA-implementation of an efficient index lookup and alignment algorithm which adapts seed-and-extend based sequence alignment to align long reads to a reference genome [7], [63]. These long reads are the output of different sequencer technologies (e.g. PacBIO [64] and Oxford Nanopore [65]) than the Illumina [66] short reads which we consider here. Long reads introduce different limitations because of their significantly higher sequence error rate.

VII. Conclusions

In this paper, we accelerated INDEL realignment with FPGAs in the cloud. IR lends itself to hardware acceleration because SIMD vectors and GPUs do not provide efficient execution, while ASICs can be prohibitively expensive and are brittle when new algorithms arise. Our work accelerates a key portion of the alignment refinement pipeline, providing a 81× speedup over GATK3 at a fraction of the cost. Leveraging FPGAs-as-a-service to deploy novel architecture and microarchitecture is synergistic with designing new domain-specific accelerators, allowing us to run Ch1–Ch22 through INDEL realignment in roughly 30 minutes for less than $1 (compared to the 42 hours needed to run GATK3).

This work faces the precision medicine immense-data challenge head on by providing a solution for time-sensitive diagnostic testing for cancer. This research is the first, as far as we know, to accelerate the alignment refinement pipeline using hardware specialization in academia, with a deployed and measured FPGA system performance close to traditional ASIC accelerators. The proposed hardware-software accelerator development framework allows for high productivity deployment of the accelerated IR system, and will ease the development of new accelerators and accelerated systems as genomic analysis pipelines change in the future.

Appendix

To understand the INDEL realignment algorithm, we provide a pictorial representation (Figure 10) and a brief glossary to familiarize the readers with genomics-specific terms used in the algorithm.

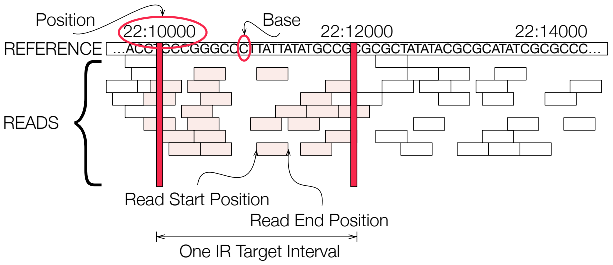


Figure 10: This figure shows an INDEL Realignment target, or IR target, from the reference position 22:10000 to 22:12000. The short reads are aligned to the reference pictorially showing how the primary alignment might have placed the reads in this region. The lightly shaded reads have either the start read position or the end read position landing inside the target region, and are considered reads for this site. To perform INDEL realignment, we slice the reference into multiple regions and perform IR on each region independently to correct systematic errors. Glossary terms *base, position, reference, read, read start/end position, and IR target* are also illustrated here.

Glossary

genomic read – A read is an output from a Next Generation Sequencing (NGS) instrument from the wet lab that determines the nucleotide sequence of a DNA or RNA sample. For this work, we use high coverage (60-65× duplicated reads per location for higher accuracy) short reads (around 250 base pairs) from Illumina sequencers [67].

base or base pair (bp) – A base is a basic unit for a genomic read, consisting of two complementary nucleotide bases. A genomic sequence is represented as a string of bases; each base is represented as one of four possible nucleotide bases – `A` for adenine, `C` for cytosine, `G` for guanine, and `T` for thymine. When a base cannot be identified clearly using the wet lab sequencing technique, `N` is used in the sequence to denote any base.

genomic position, location, or locus – A position refers to a 2-dimensional identification for where a read or a fragment of a reference starts or ends in relation to a complete genomic reference. For a human genome, we identify a sequence using its chromosome – 1 to 22, X, and Y – and where the sequence starts or ends in relation to the reference, e.g. 22:160059. After INDEL realignment, some reads are updated with more accurate start positions.

base calling – To call a base is to identify a base at a particular genomic position or location using wet lab instruments.

genomic reference – A reference is a fully sequenced and assembled genome that new sequence reads are aligned to and compared against. For this work, we use the recent release of human genome GRCh37.

quality score – A quality score is a prediction of the probability of an error in base calling [68]. For a quality score of 10, the base call accuracy is at 90%; for a quality score of 60, the base call accuracy is at 99.9999%. An industry standard *Phred Quality Score* [69], [70] is represented as a string of visible `ASCII` characters for a one-to-one mapping against a string of corresponding read bases (see IR example in Figure 4).

INDEL realignment target (IR target) or site – A target is a position interval slice in relation to the reference. Each target is generated using a start and an end reference position. Generating `t` number of IR targets is logically equivalent to slicing the reference into `t` number of slices and perform IR on each slice. All reads that overlap this region (reads that have either start or end position landing in this region) are considered reads for this site. For this work, we generate a maximum of 256 reads per target.

consensus – There are multiple ways for the reads in a given site to align to the reference and form a single genome sequence for the subject under test, i.e. the person for whom we are trying to identify genomic edits. These possibilities are called "consensuses" for this site. Consensuses are constructed using insertions and deletions present in the original alignment and reads spanning at this site given a certain heuristic. In other words, a consensus presents another way to assemble the reads using the information that's agreed upon by majority of the reads, hence the name consensus. INDEL realignment chooses the best-scoring sequence among the generated consensuses, and updates the read start positions to reflect the more accurate read alignments for the subject under test. For this work, we generate consensuses with a maximum of 2048 base pairs, and up to 32 consensuses per target.

Acknowledgment

The information, data, or work presented here was funded in part by the Advanced Research Projects Agency-Energy (ARPA-E), U.S. Department of Energy, under Award Number DE-AR0000849. Research was partially funded by ADEPT Lab industrial sponsor Intel, RISE Lab sponsor Amazon Web Services, and ADEPT Lab affiliates Google, Siemens, and SK Hynix. The views and opinions expressed here are solely those of the authors and do not necessarily state or reflect those of the United States Government or any of the sponsors. The authors would also like to thank Tae Jun Ham, Nathan Pemberton, and the anonymous reviewers for their time and valuable feedback.

References

[1] R. Nall, "Survival rates and outlook for acute myeloid leukemia," https://www.healthline.com/health/acute-myeloid-leukemia-survival-rates-outlook, October 2016.

[2] E. Papaemmanuil, M. Gerstung, L. Bullinger, V. I. Gaidzik, P. Praschka *et al.*, "Genomic classification adn prognosis in acute myeloid leukemia," *The New England Journal of Medicine*, vol. 374, June 2016.

[3] Ítalo Faria do Valle, E. Giampieri, G. Simonetti, A. Padella, M. Manfrini *et al.*, "Optimized pipeline of mutect and gatk tools to improve the detection of somatic single nucleotide polymorphisms in while-exome sequencing data," *BMC Bioinformatics*, vol. 17, November 2016.

[4] R. Gebelhoff, "Sequencing the genome creates so much data we don't know what to do with it," *The Washington Post*, July 2015.

[5] M. Zaharia, W. J. Bolosky, K. Curtis, A. Fox, D. A. Patterson *et al.*, "Faster and more accurate sequence alignment with SNAP," *CoRR*, vol. abs/1111.5572, 2011.

[6] S. Byma, S. Whitlock, L. Flueratoru, E. Tseng, C. Kozyrakis *et al.*, "Persona: A high-performance bioinformatics framework," in *Proceedings of the USENIX Annual Technical Conference*, July 2017.

[7] Y. Turakhia, K. J. Zheng, G. Bejerano, and W. J. Dally, "Darwin: A hardware-acceleration framework for genomic sequence alignment," *bioRxiv*, 2017.

[8] M. Alser, H. Hassan, H. Xin, O. Ergin, O. Mutlu *et al.*, "GateKeeper: A new hardware architecture for accelerating pre-alignment in DNA short read mapping," *Bioinformatics*, vol. 33, November 2017.

[9] Y.-T. Chen, J. Cong, Z. Fang, J. Lei, and P. Wei, "When Apache Spark meets FPGAs: A case study for next-generation DNA sequencing acceleration," in *Proceedings of the Workshop on Hot Topics in Cloud Computing (HotCloud)*, 2016.

[10] N. Ahmed, V.-M. Sima, E. Houtgast, K. Bertels, and Z. Al-Ars, "Heterogeneous hardware/software acceleration of the BWA-MEM DNA alignment algorithm," in *Proceedings of the International Conference on Computer-Aided Deign (ICCAD)*, 2015.

[11] Y.-T. Chen, J. Cong, J. Lei, and P. Wei, "A novel high-throughput acceleration engine for read alignment," in *Proceedings of the International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, May 2015.

[12] J. J. Tithi, N. C. Crago, and J. S. Emer, "Exploiting spatial architectures for edit distance algorithms," in *Proceedings of the IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2014.

[13] G. A. V. der Auwera, M. O. Carneiro, C. Hartl, R. P. G. del Angel, A. LevyMoonshine *et al.*, "From FastQ data to high-confidence variant calls: the genome analysis toolkit best practices pipeline," *Curr Protoc Bioinformatics*, vol. 43, 2013.

[14] A. McKenna, M. Hanna, E. Banks, A. Sivachenko, K. Cibulskis *et al.*, "The Genome Analysis Toolkit: a MapReduce framework for analyzing next-generation DNA sequencing data," *Genome Research*, vol. 20, 2010.

[15] Broad Institute of MIT and Harvard, "HaplotypeCaller," https://software.broadinstitute.org/gatk/documentation/tooldocs/3.8-0/org_broadinstitute_gatk_tools_walkers_haplotypecaller_HaplotypeCaller.php.

[16] M. A. DePristo, E. Banks, R. Poplin, K. V. Garimella, J. R. Maguire *et al.*, "A framework for variation discovery and genotyping using next-generation DNA sequencing data," *Nature Genetics*, vol. 43, 2011.

[17] Broad Institute of MIT and Harvard, "Broad Institute About Us: This is Broad," https://www.broadinstitute.org/about-us.

[18] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal *et al.*, "In-datacenter performance analysis of a tensor processing unit," in *Proceedings of the International Symposium on Computer Architecture (ISCA)*, 2017.

[19] A. Putnam, A. Caulfield, E. Chung, D. Chiou, K. Constantinides *et al.*, "A reconfigurable fabric for accelerating large-scale datacenter services," in *Proceedings of the International Symposium on Computer Architecture (ISCA)*, 2014.

[20] Xilinx, "Xilinx announces general availability of Virtex UltraScale+ FPGAs in Amazon EC2 F1 instances," https://www.xilinx.com/news/press/2017/xilinx-announces-general-availability-of-virtex-ultrascale-fpgas-in-amazon-ec2-f1-instances.html.

[21] Amazon, "Amazon EC2 F1 Instances," https://aws.amazon.com/ec2/instance-types/f1.

[22] R. Durbin, S. R. Eddy, A. Krogh, and G. Mitchison, *Biological Sequence Analysis: Probabilistic Models of Proteins and Nucleic Acids*. Cambridge University Press, 1998.

[23] T. F. Smith and M. S. Waterman, "Identification of common molecular subsequences," *Journal of molecular biology*, vol. 147, 1981.

[24] E. Ukkonen, "Algorithms for approximate string matching," *Information and control*, vol. 64, 1985.

[25] H. Li, "Towards better understanding of artifacts in variant calling from high-coverage samples," *Bioinformatics*, 2014.

[26] C. Xu, "A review of somatic single nucleotide variant calling algorithms for next-generation sequencing data," *Computational and Structural Biotechnology Journal*, vol. 16, 2018.

[27] Broad Institute of MIT and Harvard, "MuTect," https://software.broadinstitute.org/cancer/cga/mutect.

[28] R. Poplin, D. Newburger, J. Dijamco, N. Nguyen, D. Loy *et al.*, "Creating a universal SNP and indel variant caller with deep neural networks," *bioRxiv*, 2016.

[29] C. Saunders, W. Wong, S. Swamy, J. Becq, L. Murray *et al.*, "Strelka: accurate somatic small-variant calling from sequenced tumor-normal sample pairs." *Bioinformatics*, vol. 28, 2012.

[30] Z. Lai, A. Markovets, M. Ahdesmaki, B. Chapman, O. Hofmann *et al.*, "Vardict: a novel and versatile variant caller for next-generation sequencing in cancer research." *Nucleic Acids Res.*, vol. 44, 2016.

[31] 1000 Genomes Project Consortium *et al.*, "An integrated map of genetic variation from 1,092 human genomes," *Nature*, vol. 491, 2012.

[32] 1000 Genomes Project Consortium, "A global reference for human genetic variation," *Nature*, vol. 526, 2015.

[33] IGSR, "The international genome sample resource – providing ongoing support for the 1000 Genomes Project data," http://www.internationalgenome.org/data.

[34] GRC, "The genome reference consortium," https://www.ncbi.nlm.nih.gov/grc.

[35] H. Li, "Aligning sequence reads, clone sequences and assembly contigs with BWA-MEM," *ArXiv e-prints*, March 2013.

[36] Amazon, “F1 Clock Recipes,” https://github.com/aws/aws-fpga/blob/master/hdk/docs/clock_recipes.csv.

[37] Xilinx, “UltraScale Architecture Memory Resources: User Guide,” https://www.xilinx.com/support/documentation/user_guides/ug573-ultrascale-memory-resources.pdf, February 2018.

[38] Colin Schmidt and Berkeley Architecture Research, “RISC-V ”Rocket Chip” Tutorial,” https://riscv.org/wp-content/uploads/2015/01/riscv-rocket-chip-tutorial-bootcamp-jan2015.pdf, January 2015.

[39] J. Bachrach, H. Vo, B. Richards, Y. Lee, A. Waterman *et al.*, “Chisel: Constructing hardware in a Scala embedded language,” in *Proceedings of the Design Automation Conference (DAC)*, June 2012.

[40] K. Asanović, R. Avizienis, J. Bachrach, S. Beamer, D. Biancolin *et al.*, “The rocket chip generator,” EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2016-17, April 2016.

[41] ARM, “ARM Company Website,” https://www.arm.com.

[42] AWS, “Using AWS EDMA in C/C++ application,” https://github.com/aws/aws-fpga/tree/master/sdk/linux_kernel_drivers/edma.

[43] Xilinx, “DMA for PCI Express (PCIe) Subsystem,” https://www.xilinx.com/products/intellectual-property/pcie-dma.html.

[44] Xilinx, “AMBA AXI4 Interface Protocol,” https://www.xilinx.com/products/intellectual-property/axi.html.

[45] *TileLink Spec 1.7-draft*, https://www.sifive.com/documentation/tilelink/tilelink-spec/, SiFive, August 2017.

[46] Xilinx, “AXI4 IP,” https://www.xilinx.com/products/intellectual-property/axi/axi4_ip.html.

[47] Xilinx, “Vivado design suite,” https://www.xilinx.com/products/design-tools/vivado.html.

[48] M. Massie, F. Nothaft, C. Hartl, C. Kozanitis, A. Schumacher *et al.*, “ADAM: Genomics formats and processing patterns for cloud scale computing,” EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2013-207, December 2013.

[49] F. A. Nothaft, M. Massie, T. Danford, Z. Zhang, U. Laserson *et al.*, “Rethinking data-intensive science using scalable analytics systems,” in *Proceedings of the International Conference on Management of Data (SIGMOD)*, 2015.

[50] Xilinx, “The Xilinx SDAccel Development Environment,” https://www.xilinx.com/publications/prod_mktg/sdx/sdaccel-backgrounder.pdf.

[51] P. Klus, S. Lam, D. Lyberg, M. S. Cheung, G. Pullan *et al.*, “BarraCUDA - a fast short read sequence aligner using graphics processing units,” *BMC Research Notes*, vol. 5, January 2012.

[52] Y. Liu and B. Schmidt, “Cushaw2-gpu: Empowering faster gapped short-read alignment using gpu computing,” *IEEE Design Test*, vol. 31, February 2014.

[53] C. Lin, W. Liu, W. Yeh, L. Chang, W. W. Hwu *et al.*, “A tiling-scheme Viterbi decoder in software defined radio for GPUs,” in *2011 7th International Conference on Wireless Communications, Networking and Mobile Computing*, September 2011.

[54] N. A. Vandal and M. Savvides, “Cuda accelerated iris template matching on graphics processing units (gpus),” in *2010 Fourth IEEE International Conference on Biometrics: Theory, Applications and Systems (BTAS)*, September 2010.

[55] V. W. Lee, C. Kim, J. Chhugani, M. Deisher, D. Kim *et al.*, “Debunking the 100x gpu vs. cpu myth: An evaluation of throughput computing on cpu and gpu,” in *Proceedings of the International Symposium on Computer Architecture (ISCA)*, June 2010.

[56] Microsoft, “Microsoft computing method makes key aspect of genomic sequencing seven times faster,” https://blogs.microsoft.com/next/2016/10/18/microsoft-computing-method-makes-key-aspect-genomic-sequencing-seven-times-faster.

[57] Edico Genome, “DRAGEN Bio-IT platform,” http://www.edicogenome.com/dragen_bioit_platform.

[58] B. Harris, A. C. Jacob, J. M. Lancaster, J. Buhler, and R. D. Chamberlain, “A banded Smith-Waterman FPGA accelerator for Mercury BLASTP,” in *Proceedings of the International Conference on Field Programmable Logic and Applications (FPL)*. IEEE, 2007.

[59] I. T. Li, W. Shum, and K. Truong, “160-fold acceleration of the Smith-Waterman algorithm using a field programmable gate array (FPGA),” *BMC bioinformatics*, vol. 8, 2007.

[60] S. Huang, G. J. Manikandan, A. Ramachandran, K. Rupnow, W. H. Wen-mei *et al.*, “Hardware acceleration of the Pair-HMM algorithm for DNA variant calling,” in *Proceedings of the International Symposium on Field-Programmable Gate Arrays (ISFPGA)*. ACM, 2017.

[61] Y. Liu, D. L. Maskell, and B. Schmidt, “CUDASW++: Optimizing Smith-Waterman sequence database searches for CUDA-enabled graphics processing units,” *BMC research notes*, vol. 2, 2009.

[62] T. Rognes, “Faster Smith-Waterman database searches with inter-sequence SIMD parallelisation,” *BMC bioinformatics*, vol. 12, 2011.

[63] Y. Turakhia, G. Bejerano, and W. J. Dally, “Darwin: A genomics co-processor provides up to 15,000x acceleration on long read assembly,” *asplos*, April 2018.

[64] PACBIO, “PACBIO,” https://www.pacb.com.

[65] Oxford Nanopore Technologies, “Oxford Nanopore Technologies,” https://nanoporetech.com/.

[66] Illumina, “Illumina,” https://www.illumina.com.

[67] Illumina, “An introduction to next-generation sequencing technology,” https://www.illumina.com/content/dam/illumina-marketing/documents/products/illumina_sequencing_introduction.pdf.

[68] Illumina, “Understanding Illumina Quality Scores,” https://www.illumina.com/documents/products/technotes/technote_understanding_quality_scores.pdf.

[69] B. Ewing, L. Hillier, M. C. Wendl, and P. Green, “Base-calling of automated sequencer traces using Phred,” *Genome Research*, vol. 8, February 1998.

[70] P. J. Cock, C. J. Fields, N. Goto, M. L. Heuer, and P. M. Rice, “The Sanger FASTQ file format for sequences with quality scores, and the Solexa/Illumina FASTQ variants,” *Nucleic Acids Research*, vol. 38, December 2009.