Co-Design for Energy Efficient and Fast Genomic Search: Interleaved Bloom Filter on FPGA

Marius Knaust
Zuse Institute Berlin
Berlin, Germany
knaust@zib.de

Enrico Seiler
Freie Universität Berlin
Berlin, Germany
enrico.seiler@fu-berlin.de

Knut Reinert
Freie Universität Berlin
Berlin, Germany
knut.reinert@fu-berlin.de

Thomas Steinke
Zuse Institute Berlin
Berlin, Germany
steinke@zib.de

ABSTRACT

Next-Generation Sequencing technologies generate a vast and exponentially increasing amount of sequence data. The Interleaved Bloom Filter (IBF) is a novel indexing data structure which is state-of-the-art for distributing approximate queries with an in-memory data structure. With it, a main task of sequence analysis pipelines, (approximately) searching large reference data sets for sequencing reads or short sequence patterns like genes, can be significantly accelerated. To meet performance and energy-efficiency requirements, we chose a co-design approach of the IBF data structure on the FPGA platform. Further, our OpenCL-based implementation allows a seamless integration into the widely used SeqAn C++ library for biological sequence analysis. Our algorithmic design and optimization strategy takes advantage of FPGA-specific features like shift register and the parallelization potential of many bitwise operations. We designed a well-chosen schema to partition data across the different memory domains on the FPGA platform using the Shared Virtual Memory concept. We can demonstrate significant improvements in energy efficiency of up to 19× and in performance of up to 5.6×, respectively, compared to a well-tuned, multithreaded CPU reference.

CCS CONCEPTS

• **Computer systems organization** → **Reconfigurable computing**; • **Applied computing** → **Bioinformatics**; **Computational genomics**.

KEYWORDS

sequence alignment, indexing, FPGA, energy efficiency, performance

ACM Reference Format:
Marius Knaust, Enrico Seiler, Knut Reinert, and Thomas Steinke. 2022. Co-Design for Energy Efficient and Fast Genomic Search: Interleaved Bloom Filter on FPGA. In *Proceedings of the 2022 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA '22), February 27-March 1, 2022, Virtual Event, CA, USA.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3490422.3502366

FPGA '22, February 27-March 1, 2022, Virtual Event, CA, USA.

ACM ISBN 978-1-4503-9149-8/22/02.
https://doi.org/10.1145/3490422.3502366

1 INTRODUCTION

Following the sequencing of the human genome [10, 27], genomic analysis has come a long way. The recent improvements of sequencing technologies, commonly subsumed under the term NGS (Next Generation Sequencing) or 3rd (and 4th) generation sequencing, have triggered incredible innovative diagnosis and treatments in biomedicine, but also tremendously increased the sequencing throughput. Within 10 years it rose from 21 billion base pairs [10, 27] collected over months to about 400 billion base pairs per day on a single machine (current throughput of Illumina's HiSeq 4000).

The costs for producing one million base pairs could also be reduced from hundreds of thousands of dollars to a few cents. As a result of this dramatic development, the number of new data submissions, generated by various biotechnological protocols (ChIP-Seq, RNA-Seq, assembly etc.), to genomic databases has grown dramatically and is expected to continue to increase faster than the cost per capacity of storage devices will decrease. This poses challenges for the existing sequence analysis pipelines. One of the main tasks of such pipelines is to (approximately) search large reference data sets for sequencing reads or short sequence patterns like genes. Hence, researchers had to develop novel indexing data structures such as the *Interleaved Bloom Filter* (IBF) [7] and, based on this, an extension with winnowing minimizers and probabilistic thresholding called *Raptor* [23] which is currently the state-of-the-art for distributing approximate queries with an in-memory data structure. The CPU-based IBF implementation can distribute 10 million NGS queries for combined texts of hundreds of Gigabytes in only a few seconds.

Here, we see the opportunity to further improve this algorithmic development with the unique characteristics of FPGAs, both in terms of performance and energy efficiency. The minimizer computation is ideally suited to use shift-registers, a pattern that translates directly to the hardware of the FPGA, because they follow a sliding window approach. The IBF computations perform many parallel bitwise operations that can benefit from the high parallelism of FPGAs at this level. In addition, genomics applications are generally very well suited for non-standard data types, such as a 2 bit representations for DNA without ambiguity characters.

The **contributions of our work** are the following:

- We present a hardware/software co-design approach for a first implementation of the novel IBF data structure on FPGA which results in a 5.6× speedup compared to the highly-tuned CPU implementation in the SeqAn C++ library [20].
- We can demonstrate a 19× improvement in energy efficiency which is in particular important in institutional and data center environments to meet power restrictions.
- We demonstrate today's performance capabilities within a high-level language (OpenCL) approach for the implementation on FPGA. By directly mapping sliding windows in the algorithm to shift registers in hardware, using non-standard data bit-widths, and taking advantage of the parallel-processing capabilities on bit level on FPGAs, we are able to turn a compute-bound problem into a memory- (latency) bound problem.
- Our implementation can be transparently used in the SeqAn library, and, in conjunction with the demonstrated performance and energy efficiency, we contribute to enabling FPGAs in today's large-volume sequence searches for genomic but also other texts.

2 RELATED WORK

In this section, we highlight recent related work in the (bioinformatics) algorithm domain as well as implementations of approximate genomic searches on FPGAs.

The problem of approximately searching queries in ultra-large databases has been recently addressed by several groups focusing on different applications, all of which use methods based on the k-mer content of the databases. In the field of alignment-free metagenomic analysis, which focuses on k-mer based analysis, the size of the data is likewise slowly becoming prohibitive. For example, Kraken [28] requires 147 GiB of RAM to index 380 GBases. For the analysis of RNA-Seq data, some groups aimed to search the raw files directly for a series of transcripts ([25] and shortly afterwards [26]), [3]). They propose novel solutions to the problem of searching a transcript of interest in all relevant RNA-Seq experiments. The runtimes were initially significantly improved by the Patro group with the tool Mantis in [18] and by the Iqbal group with COBS [3]. This year, the Reinert lab introduced the IBF [23], which has proven to be a significant step towards a very time and space efficient in-memory data structure for preprocessing approximate sequence queries, which opens up many possible applications. It improves in runtime by a factor of 12-144 over its competitors COBS and Mantis.

Recent work in the area of approximate genomic searching including short read alignment on FPGA are presented in comprehensive surveys [17, 21, 22], and a summary in [15]. Most recent FPGA-based accelerator solutions are based on the FM-index data structure. Ng et al. [16] implemented a two-stage alignment architecture on FPGA that is similar to the seed-and-extend model adopted by Bowtie2, and achieved an overall 2× speedup. For the construction of an FM-index, Chen et al. [5] propose an FPGA-based implementation of the SAII algorithm with no memory overhead. As far as we know, no FPGA or other accelerator implementations of Mantis and COBS are currently available. It should be noted that efforts tackling the memory access issue are presented, for example, with the GRIM-Filter on HBM [11], and ALPHA filter on PiM architecture [8], both of which show significant performance improvements. Also, pure k-mer counting was accelerated on the FPGA in [14] by a factor of 13 compared to CPU-based solutions. Further, Sireesha and Roopa provided in [24] an implementation of standard Bloom filters on an FPGA for key value stores. While related, the above implementations do not address the problem presented in this work.

Our work differs from previous ones as we propose a hardware/software co-design for the implementation of the novel IBF data structure on reconfigurable hardware, chose an implementation approach which allows a seamless drop-in replacement into the existing SeqAn library, and thus ensure the same valid biological results as the CPU reference.

3 BACKGROUND

The following section contains a brief introduction to the IBF data structure, details can be found in [23].

The IBF data structure addresses the problem of indexing and querying large collections of data. While the data consist of long sequences of (DNA) characters, those sequences are often tokenized by generating all substrings of length k, also called the k-mers of a sequence. Then, the input data can be seen as a set of sets of k-mers with which the user usually connects a semantic meaning. For example, the user could define about 20000 sets of bacterial genomes where each set contains the k-mers of genomes of all the subspecies of a particular species. Or the user might want to query a collection of 3000 RNA-Seq files that contain NGS reads. Then a set contains the k-mers of all the reads in a file.

The IBF stores a *representative* transformation of the k-mer content of the database (for instance, many thousands microbial genomes). A k-mer in this context is a substring of string of length k, e.g. $T = AGGCT$ is a string of length 5 containing two 4-mers $AGGC$ and $GGCT$. The database consists of bins B_i, typically a few hundred to a few thousand, for example, a bin for all genomes of the subspecies of a certain bacterial species.

The term *representative* indicates that the k-mer content could be transformed by a function which reduces its size and distribution (for example, using winnowing minimizers on the text and its reverse complement, or using gapped k-mers). We currently use ungapped (w, k)-*minimizers* to compute representative k-mers (see also [6]). A (w, k)-minimizer is essentially the lexicographically minimal k-mer of all k-mers and their reverse complements in a window of size w. For example, choosing $w = 5$ and $k = 4$ for the string $S = ATTACGTA$ yields for the first window of S, $ATTAC$, the two 4-mers $ATTA$ and $TTAC$. The reverse complement of the first window, $GTAAT$, yields $GTAA$ and $TAAT$. In total, there are four 4-mers $\{ATTA, TTAC, GTAA, TAAT\}$. Hence, the minimizer for the first window is $ATTA$. To determine all minimizers of S, the window is shifted one position to the right after the current window is processed. (w, k)-minimizers can be efficiently computed using a rolling hash function.

The same transformation is applied to the k-mers of the query upon lookup. The parameter w is set depending on the length of the query and the maximum of allowed errors e and is assumed

to be given in this work. The IBF is then used to retrieve *binning bitvectors* indicating whether a representative k-mer is in a bin.

As such, the IBF is a combination of b standard Bloom filters [4] in an interleaved scheme. A Bloom filter is a bitvector of size n and a set of h hash functions that map a key value, in our case a representative k-mer, to one of the bit positions. A value is present in the Bloom filter if all h positions return a 1. Note that a Bloom filter can give a false positive answer. However, if the Bloom filter size is large enough, the probability of a false positive answer is low. A Bloom filter of size n bits with h different hash functions and m inserted elements has a probability for giving a false positive answer of:

$$p_{fp} = \left(1 - \left(1 - \frac{1}{n}\right)^{h \cdot m}\right)^{h}$$

For this reason, sufficient space needs to be allocated such that p_{fp} does not become too large. Still, the problem of using a simple Bloom filter is that it does not point directly to the binning bitvectors, i.e., it only answers the set membership question for a single bin. To alleviate the problem, b Bloom filters (one for each bin) with identical hash functions are used and their bitvectors are interleaved. In other words, in an IBF each bit in the normal Bloom filter is replaced by a (sub-) bitvector of size b, where the i-th bit "belongs" to the Bloom filter for bin B_i. This results in a total size of $b \cdot n$. When inserting a k-mer from bin B_i into the IBF, all h hash functions are computed, each pointing to the position of the block in which the sub-bitvector is stored. For each of the sub-bitvectors, the i-th bit from the respective beginning is set. Hence, b Bloom filters are interleaved in a way that allows the binning bitvectors for each of the h hash functions to be retrieved in a single burst operation. When querying which bins contain a k-mer, the h sub-bitvectors are retrieved and a bitwise AND-operation is applied, which then results in the desired binning bitvector indicating the membership of the k-mer in the bins (see Figure 1). This is much faster than individually querying b standard Bloom filters to retrieve the binning bitvector.

For the approximate search of a query P, the binning bitvectors of all representative k-mers in the query are combined into a *counting vector* and the membership of a query in a bin is determined by applying an appropriate threshold (see [23]). This approach is depicted in Figure 2.

In summary, the main computational steps that have to implemented on the FPGA are:

(1) The minimizer computation: Compute the lexicographical minimum for all k-mers and their reverse complement in a window of size w.
(2) The IBF computations: Compute h hash functions for the minimizers, the bitwise AND of the h binning bitvectors, increment counters, and test for exceeded threshold.)

4 DESIGN CONSIDERATIONS FOR THE FPGA IMPLEMENTATION

The goal of the FPGA-accelerated IBF implementation is to be a drop-in replacement for the existing CPU implementation of the pattern look-up described in the previous section. The host submits patterns to the accelerator, while in return it receives the bin associations

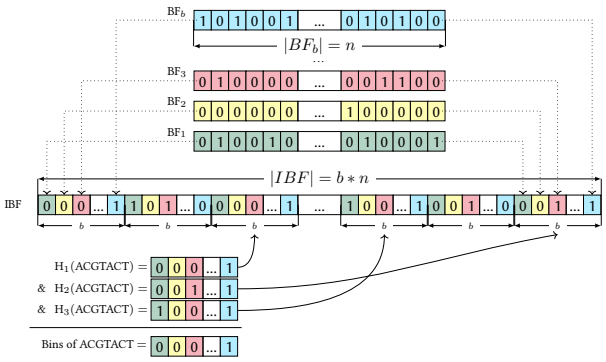

Figure 1: Example of an IBF. Differently colored Bloom filters of length n for the b bins are shown at the top. The individual Bloom filters are interleaved to obtain an IBF of size $b \times n$. In the example, 3 positions are retrieved for a k-mer (ACGTACT) using 3 different hash functions. The corresponding sub-bitvectors are combined with a bitwise AND-operation, which results in the required binning bitvector (Figure is from [7]).

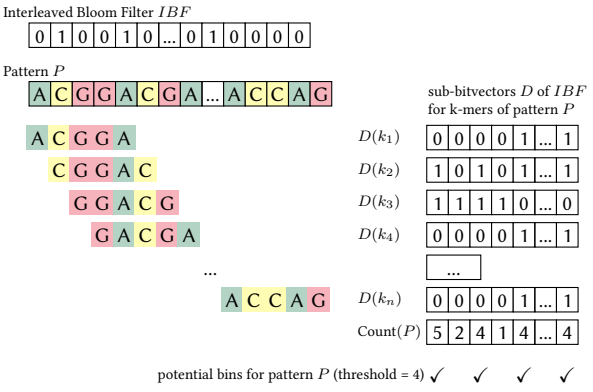

Figure 2: For each k-mer k_i generated from a pattern P, the binning bitvectors $D(k_i)$ are extracted. $D(k_i)$ represents the bins containing the k-mer k_i. For all bits in $D(k_i)$ which are set to 1, the counter of the corresponding bin is incremented. Bins with a counter greater than or equal to the threshold (in this case 4) need to be validated for P (Figure is from [7]).

as a bit mask. Like Seiler et al.[23], it uses minimizers to reduce the number of k-mers to be queried and thus the number of costly memory accesses.

We decided on the *Intel FPGA SDK for OpenCL* version 2021.3 as the implementation environment, as it offers a high-level programming model with an acceptable overhead and encapsulates the entire host interaction in a well-known API, which allowed us to focus on the algorithmic optimizations of the problem. The CPU emulation enabled quick functional tests after performance optimizations. Most of these optimizations were made based on

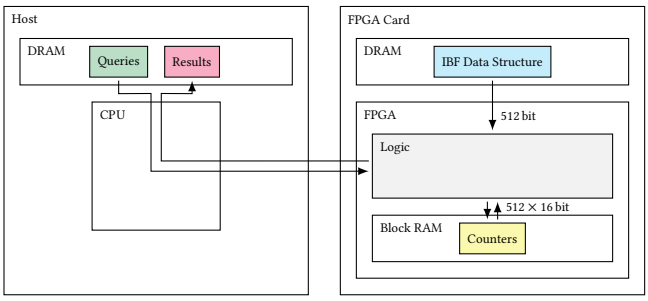

Figure 3: A simplified overview of the memory partitioning of the most important data structures across the memory hierarchy of the FPGA card and the host system. With the queries and the results placed on the host and accessed directly via the PCIe connection. This leaves the DRAM directly attached to the FPGA solely to IBF data structure, which enables maximum performance, since it is accessed by a read-only 512 bit burst-coalesced load-store unit. In addition, the counters for a higher number of bins are stored in the BRAM and are connected via a wide read and write port to serve the 16 bit counters of all 512 bins that are processed simultaneously.

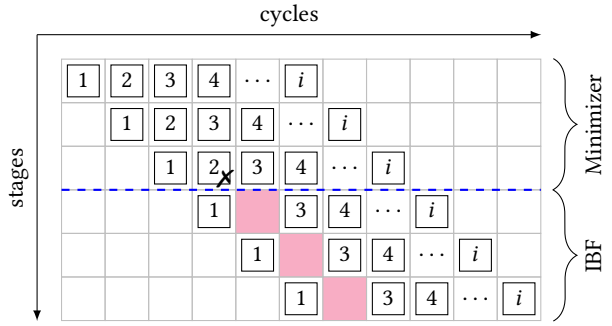

Figure 4: Pipeline bubbles (highlighted in red) in the IBF computation (bottom) when it is combined with the minimizer computation (top) in a single deep hypothetical pipeline. In the example shown, a pipeline bubble is generated if the second iteration is skipped after the minimizer computation.

the detailed reports of the High-Level Synthesis (HLS). Thus, the time-consuming bitstream generation was only necessary for the final evaluation and the possibility of the automatic insertion of profiling counters into these led to the detection of bottlenecks induced by memory interaction. Further, this approach helps our intention to integrate our work seamlessly into to the SeqAn library for broader impact.

The various data structures were carefully partitioned over the entire memory hierarchy of the FPGA card and the host system to ensure maximum performance (Figure 3). The large prebuilt IBF data structure is stored on the DRAM of the FPGA, interleaved across all banks so that it can be accessed concurrently.

We opted for the Board Support Package (BSP) that supports Shared Virtual Memory (SVM), which allows the accelerator to access host memory directly via the PCIe connection. This opened up the possibility for us to leave data that is only accessed once (either by a read or a write access) on the host and to transfer it directly without a detour over the FPGA-attached DRAM (hereinafter simply referred to as DRAM), which as a result reduces the load on it. In this particular case, this applies to the majority of the most important data structures apart from the prebuilt IBF data structure, such as the input pattern and the resulting bit set. Since the IBF data structure remains the only major data structure located on the DRAM and (after the initial transfer) is only accessed for reading, low latencies and maximum burst rates can be achieved (which were confirmed by profiling).

In order to enable multi-FPGA support in future work, it should be possible to adapt the partitioned version of the IBF [7] in which the indexing structure is broken down into several partitions in order to distribute it.

For the remainder of this section, we will first focus on the accelerator side and later on the host side, which involves the interaction between the two.

4.1 Implementation on the FPGA Accelerator

For the FPGA accelerator implementation, we use single work-item kernels, as recommended by Intel in their programming guide [9]. In order to enable static optimizations such as shift-registers and since certain parameters do not vary for inputs of the same type, we keep the k-mer size, window size, and number of bins compile time constant and have several bitstreams available at runtime (Section 4.2). Since only a few bits are necessary to encode base pairs, for example DNA without ambiguity characters requires 2 bit, we heavily use data types with custom bit-width throughout our design. We opted out of Intel's hyper-optimized handshaking because it does not allow for query prefetching and other memory access optimizations that we favor in our design.

At first glance, it seems favorable to combine the minimizer and IBF computations on the FPGA, as this would enable deeper pipelining and thus make better use of the parallel resources, even if it is not possible to express explicit parallelism. However, the minimizer computation skips elements if they are redundant to their predecessor, this in turn would lead to bubbles in the rest of the pipeline that extends over the entire IBF computation (Figure 4). As a consequence, we decided to decouple the minimizer and IBF computations by splitting them into two separate kernels that run in parallel and communicate via a FIFO buffer. In order to realize this, we used the Intel-specific *Intel FPGA SDK for OpenCL Channels Extension*. For reasons of compatibility with OpenCL SDKs from other vendors, it should be possible to replace its use with pipes from OpenCL 2.0.

The IBF kernel ends up operating slower than the minimizer kernel due to costly memory interactions (Section 5.2.1). As a result, the minimizer kernel usually stalls waiting for the IBF kernel to free space in the buffer. This is not a disadvantage in our particular configuration, since the memory interaction dominates the computation anyway and there is therefore no risk of the buffer running empty.

To saturate all given DRAM memory banks, the IBF data structure is interleaved across all of them, and we replicate the kernel pair of minimizer and IBF computations, each with its own FIFO

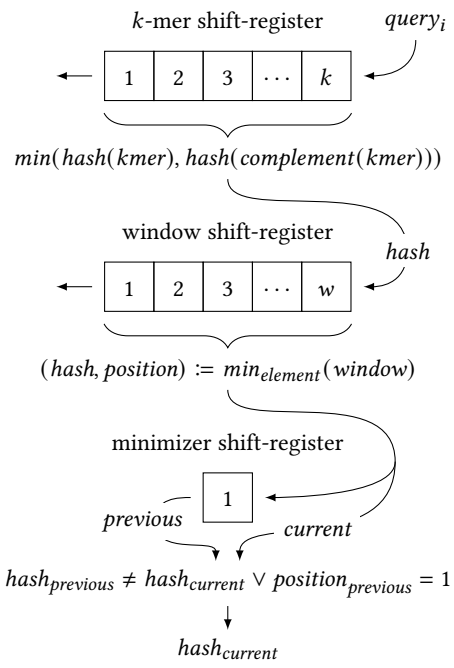

Figure 5: Composition of shift-registers in the minimizer computation. Starting with a shift-registers containing the current k-mer, followed by one for the hash values of the current window, and ending with a single register containing the previous minimizer to decide whether to skip the current minimizer.

buffer. The parallelization is naive as each pair processes separate queries (Section 4.2). The number of pairs is determined experimentally, since the higher resource requirements also affect the performance of the configuration.

4.1.1 Minimizer Kernel. The minimizer kernel heavily relies on three chained shift-registers to buffer data and reduce redundant memory interactions. With each iteration over the query elements, all shift registers are advanced by one position and provide data to their successors (Figure 5).

- **The first shift-register** contains the current k-mer, which is updated by the latest query element. Based on its content, the hash of the k-mer and that of its reverse complement is calculated; the smaller of the two is passed on to the next shift-register.
- **The second shift-register** contains all the representative hashes of the current window. The first occurrence of the minimizing hash (together with its position in the window) is determined for each window and fed into the last shift-register.
- **The third shift-register** contains only a single element, the previous hash. The current hash is ignored if its value is the same as the predecessor's and the predecessor itself is still valid (not moved out of the sliding window), otherwise it is passed on to the IBF computation.

Since the minimizer computation and the IBF computation are carried out separately from each other, the minimizer computation has to mark the last minimizer of the query so that the decoupled IBF computation knows when to stop and transmit its results to the host. Because the minimizer computation can not tell whether the current element will be the last minimizer or not, the previous minimizer is only passed when a new one is determined or the last query element is reached. This is achieved by leveraging the last shift register, which buffers the previous minimizer anyway.

4.1.2 IBF Kernel. The IBF kernel processes incoming minimizer hashes as long as none is marked as the last element of the query. For smaller bin sizes, it operates as expected: First, it reads the binning bitvectors of the IBF for the corresponding bin of each hash function from the DRAM; then all binning bitvectors are combined via a binary `AND`-operation. Afterwards, a separate counter for each set bit in the binning bitvector is incremented. As a result, a bit mask is created in which each bit is set based on the associated counter exceeding a provided threshold, which depends on the IBF's parameters.

The main strength of the FPGA implementation is that all the operations on each bit of the bin can be carried out in parallel, such as the `AND`-operation, but especially the counter increments and the threshold checks. This would not be possible on the CPU even when using AVX-512, as it is limited to 32 bins at the same time with 16 bit counters.

The original implementation was relying on a modulo operation to map a hash to a bin. The realization of the normal modulo operation requires a division, which is quite expensive to implement on the FPGA. We have therefore decided to replace the modulo operation with an alternative introduced in [12], which is based on a wide multiplication followed by a shift operation, both of which are manageable on the FPGA. It has been proven that the values are mapped equally well to the specified range, which is the important requirement for the IBF.

We have also decided to give each access point to the IBF data structure based on a different hash function its own load-store unit (LSU). If the data is stored on different memory banks, this allows them to operate simultaneously. But more importantly, each LSU has its own independent burst buffer so that they do not interfere with one another in this regard.

Optimizations for larger numbers of bins. The implementation for larger number of bins is more complicated, as two challenges arise: First, for a larger number of bins, the binning bitvectors exceed the width of the internal memory interface (512 bit in our case). Second, the array of counters becomes very large, so it is no longer feasible to keep them in registers, and it has to be stored in block RAM (BRAM). Both problems require special handling to achieve good performance.

Exceeding the width of the internal memory interface leads to a stall of the pipeline, since it takes more than one cycle to fulfill the memory request. Hence, it is better to divide the memory access into several chunks with the width of the memory interface and to overlap the individual requests with the computation on the previous chunk. This optimization avoids the stalls and at the same time reduces the area demand, since the computation is, in parts, carried out serially.

Since, with large bin sizes, the counters have to be stored in BRAM, it is important to configure this memory correctly so that it meets the requirements placed on it and at the same time remains within the hardware limits. We decided to initialize the counters lazily the first time they are accessed, as this reduces the requirement to just a single read and a single write port that matches the underlying hardware. This makes it sufficient to have a single bank non-replicated BRAM configuration. However, the bank width is chosen to be wide enough to allow access to all counters of a chunk at the same time in order to enable parallel computation (see Figure 3). Here the chunking plays into our hands, as it limits the required bank width.

Another positive side effect of the chunking is that it solves another problem that has been introduced by placing the counters in BRAM, a memory dependency. BRAM has no unpredictable large latency like DRAM, but it still has a small latency between consecutive reads and writes to the same location, unlike registers. This would lead to a memory dependency in our design as the counters are incremented with each iteration. However, since each chunk accesses a different set of counters, the memory dependency is resolved as long as there are more chunks than the number of cycles imposed by the latency.

For smaller IBF sizes, it might be feasible to use modern FPGAs with High Bandwidth Memory (HBM) to store the IBF data structure. Since the HBM is connected with a wider memory interface, more data can be served at a time, which should lift the current 512 bit limit and unleash more parallelization potential. Furthermore, a hierarchical version of the IBF is under development, dividing the structure into separate levels, which are typical smaller. This may make it possible to preload the individual levels from the DRAM into HBM.

4.2 Host Part of the Implementation

Since certain parameters (k-mer and window size as well as the number of hash functions and bins) are statically compiled into our FPGA design in order to enable further optimizations, we maintain a bitstream library for the common parameters sets. During initialization, the host configures the FPGA with the corresponding bitstream for the specified parameters, and at the same time loads the pre-built IBF data structure from the filesystem into the accelerator's DRAM. Depending on the size of the IBF data structure, one or the other dominates (Figure 6). After the initialization is initiated (but not necessary completed), it is ready to accept requests.

It is not feasible to process individual queries, especially small ones, because the overhead involved in the transfer of the data to the accelerator and starting the kernels is too big. As a consequence, the host maintains a buffer in which it gathers queries and submits them to the accelerator in batches. At the same time, this enables the multiple kernel pairs (Section 4.1) to process the queries in parallel, as each simply takes on an equal share. In order to overlap I/O and computation, the host actually maintains two of these buffers in a double-buffer pattern. While the accelerator performs the computations on one buffer, the other is being filled with new data by the host. Since both the input and output are double buffered, it is possible for the host to carry out further computations based on the results already computed in parallel to the processing of the remaining queries by the accelerator.

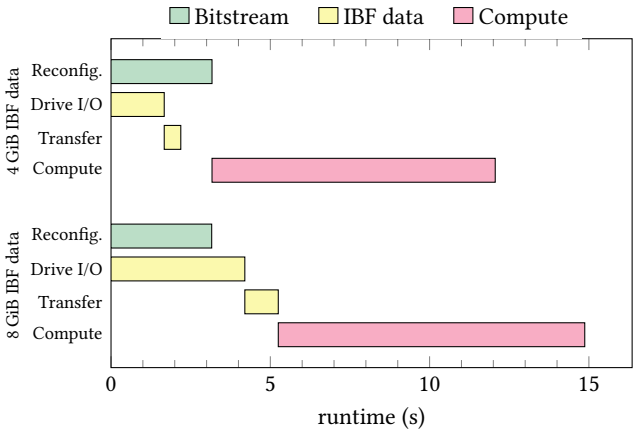

Figure 6: The simplified execution timeline for 10 million queries and a configuration with 8912 bins, for both a 4 GiB and a 8 GiB IBF data structure. Since the reconfiguration of the FPGA and the initialization of the IBF data structure (reading it from filesystem and the transfer to the FPGA's DRAM) take place in parallel, one or the other dominates the initialization phase, depending on the size of the IBF data structure.

5 RESULTS

The FPGA implementation was evaluated on an Intel FPGA PAC D5005 card with an Intel Stratix 10 SX FPGA (14 nm lithography, 2 753 000 logic elements, 244 Mbit on-chip memory, 11 520 DSP blocks) and a total of 32 GB (4 banks of 8 GB each) external DDR4 DRAM. The card is attached to the host system via PCIe 3, even though our measurements seem to indicate that the BSP is only using 8 of the 16 PCIe lanes. The FPGA host system comprises two Intel Xeon Gold 6246 CPUs with 384 GiB of main memory. The system runs with CentOS 7.8 and kernel version 3.10.0., and Board Management Controller version 2.0.12 is in effect.

The CPU reference implementation [23] is benchmarked on dual socket system with Intel Xeon Gold 6248 CPUs (14 nm lithography) and 1 TiB main memory. The CPU reference benchmarks run with 32 threads on this system.

A set of approximately 10 million artificial DNA sequence patterns with a read length of 100 base pairs was used for the first measurement. In addition, we conducted experiments with a growing number of patterns reaching from 10 to 50 million. With regard to the parameters, we have chosen a k-mer size of 19, a window size of 23, and a IBF data structure size of 4 GiB as a typical configuration for this type of queries. For the number of bins, we have selected 64 at the lower end (such a value is for example used in [7] to distribute read mapping jobs) and 8192 at the upper end (such a value would be used for metagenomics analyses like in [19] to represent a range encountered in common use cases and cover both

Table 1: The utilization of resources of the Intel Stratix 10 SX FPGA for the two designs used in the benchmarks. The design for many bins (8192 bins) uses 4 pairs of minimizer and IBF kernels to achieve the best performance. In contrast to that, only 2 pairs are sufficient when the strategy for few bins (64 bins) is employed.

	Logic Utilization		BRAM	DSP
	ALUTs	Registers		
Device	933 120		11 721	5760
	1 866 240	3 732 480		
8192 bins	391 505 (42 %)		5622 (48 %)	132 (2 %)
(4 pairs)	322 215	773 426		
64 bins	299 362 (32 %)		2722 (23 %)	70 (1 %)
(2 pairs)	242 906	484 191		

implementations of the IBF kernel specialized in the number of bins (Section 4.1.2).

Each benchmark was repeated 120 times, aiming for a confidence level of 99 % for the given precision of the stated mean values.

5.1 FPGA Design Properties

The FPGA design reaches frequencies of around 300 MHz in the 8192 bin design and around 200 MHz in the 64 bin version, depending on the number of minimizer and IBF kernel pairs used. For the specific configuration of the benchmarks, the best performance can be observed with 4 and 2 kernel pairs for 8192 and 64 bins, respectively, leading to frequencies of 297 MHz and 211 MHz. Thanks to the manually resolved memory dependency in the IBF kernel (Section 4.1.2), the High-Level Synthesis (HLS) reports only low Initiation Intervals (II) for all critical pipelined loops.

5.1.1 Resource Utilization. With the OpenCL-based high-level language approach, a number of optimization capabilities are available to the developer that directly influence the resource usage. For example, enforcing loop unrolling through directives, the way the Load Store Units are used, e.g., with/without cache, the replication of compute units (here the kernel pairs) which multiplies logic and BRAM resources, and the implementation of arithmetic operations via DSPs (here the modulo operation) impact the resource utilization.

Our achieved resource utilization for both small and large numbers of bins is summarized in Table 1. The numbers shown include the static part given by the BSP (e.g. PCIe link, memory controller, partial reconfiguration) and the freely available resources consumed by the user logic. The resource utilization of the user logic scales roughly linear with the number of kernel pairs placed on the FPGA. Since the implementation for large number of bins differs from the more basic implementation for a few bins (Section 4.1.2), they cannot be directly compared.

5.2 Runtime Performance

Table 2 summarizes the performance and energy consumption data (see 5.3) of our work together with some recent data for FPGA implementations of similar algorithms.

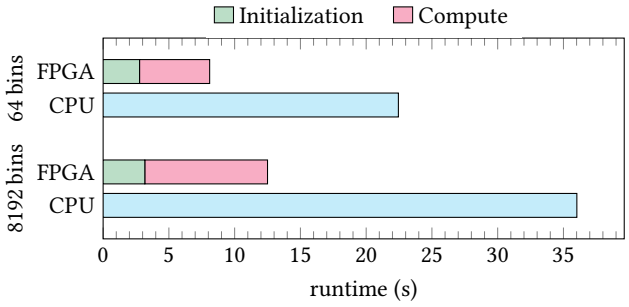

Figure 7: The FPGA's runtime compared with the multi-threaded CPU reference implementation for approximately 10 million queries (reads), for both few (64) and many (8192) bins. The runtime of the FPGA implementation is further divided into an initialization and a computation phase, the initialization loses relevance with larger query volumes and thus leads to an even higher speed-up factor.

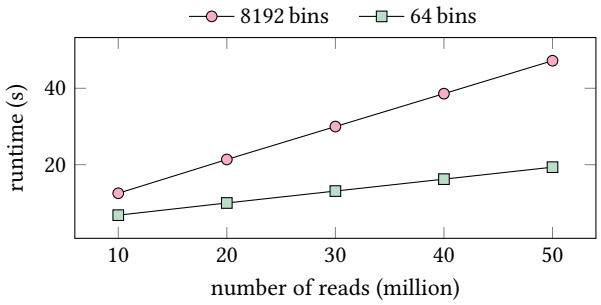

Figure 8: The runtime on our FPGA platform plotted for an increasing amount of reads, for both few (64) and many (8192) bins. A linear scaling can be observed.

For 10 million sequence patterns (reads), we measure for the total runtime of the 8192 bin configuration on the FPGA accelerated implementation an average of 12.51 s (standard deviation (SD) = 0.05 s) , which is a 2.88-fold increase over the reference implementation with a runtime of 36.02 s (Figure 7). At 64 bins we can observe an even higher speed-up factor of 3.30 × with an average runtime of 6.79 s (SD = 0.04 s) versus a runtime of 22.45 s. For few bins, all bins can be fetched from memory at the same time and the array of counters is small enough to fit in registers. With an increasing amount of reads, the runtime scales linearly (Figure 8).

For 50 million reads, we measure the best speedups of the FPGA accelerator implementation over the CPU reference in our study. The FPGA configurations with 1892 and 64 bins is 3.4 × and 5.6 ×, respectively, faster than the CPU version. The total runtimes of the 8192 bin configurations are on average 47.21 s (SD = 0.15 s) on the FPGA accelerator and 162.2 s on CPU. Using the 64 bin configuration, the total runtimes are 19.35 s (SD = 0.055 s) on FPGA and 108.00 s on CPU, respectively.

It should be noted that these measurements include the initialization phase. For both systems, this is the I/O to read the prebuilt

IBF data structure from the filesystem. The measurements relating to the FPGA implementation also include its transfer to the DRAM of the FPGA and the reconfiguration of the FPGA loading the bitstream. Since these initialization procedures only have to be carried out once, regardless of the number of queries, the increase in performance of the pure computational part is even higher than the factors mentioned above. Namely, an estimated (because of parallel flows during initialization in both implementations) factor of 3.80 for the 8192 bin configuration and even 5.85 for the 64 bin configuration while processing 10 million reads.

Compared with recent FPGA implementations of the FM-index structure (Table 2), our IBF data structure implementation is significantly faster without any limitations of the biological relevance of its results [23]. Moreover, our IBF data structure implementation on CPU is competitive to some FPGA accelerated FM-index implementations.

This clearly shows the usefulness of the IBF as prefilter for databases that are so large that a single FM-index is too costly to compute. In such cases the IBF can effectively distribute the queries to many smaller FM-indices as was demonstrated in [7].

5.2.1 Limitations. When instrumenting the LSUs with profiling counters, it becomes apparent that the memory access of the IBF data structure in DRAM causes the pipeline to a stall about 30 % of the time for the 8192 bin configuration. This value is rather modest because it is lowered by the bursts. This becomes visible with the 64 bin configuration, since it does not use bursts and leads to a stall rate of up to 85 %. The use of a caching LSU, which temporarily stores memory requests that are accessed in the BRAM, is only of little help, especially for bigger IBF data structure sizes, since the hash-based accesses occur randomly across the whole IBF data structure. This is an inherent problem with Bloom filters [13] because they rely on this randomness. Nevertheless, this indicates that we have turned a previously compute-bound problem into a memory latency-bound problem.

5.3 Energy Consumption

In addition to runtime performance, energy consumption has become an important factor for data centers in recent years. The FPGA-based implementation has a great advantage in this area, partly because of the reduced runtime, but also because of the less generic, more problem-adapted hardware of the accelerator.

To measure the power consumption of our Intel FPGA card, we use the *Open Programmable Acceleration Engine* (OPAE) SDK version 1.1.4 to access the *Platform Descriptor Records* (PDR), which contain information about the card's power subsystem. The D5005 card has two relevant entries, one for the PCIe and one for the auxiliary connector, which can be aggregated to the total power consumption of the card. We then integrate the power draw over the runtime to determine the total energy consumption of the application (Figure 9).

The power consumption of the CPU implementation is measured with the *Running Average Power Limit* (RAPL) counters of modern Intel CPUs. We use *perf* to set the measurements into relation to the runtime of the applications and add up the *package* and *DRAM* zone to use them as the total energy demand.

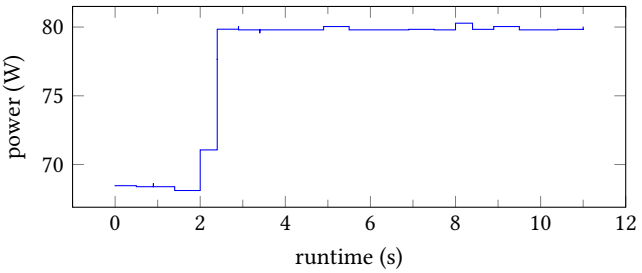

Figure 9: The power draw of the FPGA card plotted over the runtime of the 8192 bin configuration. Starting with a reconfigured FPGA at idle while the host reads the precomputed data structure from the filesystem, followed by a brief (approximate half a second) increase while the data structure is transferred to the FPGA accelerator, and finished with the compute phase that requires the most power. The plotted power measurements can be integrated over the runtime to obtain an estimate of the total energy consumption.

For 10 and 50 million reads, the 8192 bin configuration on the FPGA card consumes on average 888 J and 3591 J (SD = 13.8 J, 22.6 J), respectively, making it about 11 times more energy efficient than the CPU reference implementation with its energy consumption of 10 412.36 J and 41 442 J. The improvement in the 64 bin configuration is superior again, as is with the runtime performance. It reaches a factor of up to 19 × with an average consumption of 401 J and 1170 J (SD = 14.36 J, 14.0 J) versus 5223 J and 22 600 J on CPU. This excludes the power consumption of its host system, since it only performs I/O and would be free for other computational workloads based on the results (Section 4.2). If there is no demand for further processing and its only purpose is to host the FPGA card, the host could be configured to have a very low energy footprint. The power consumption of the FPGA card might be reduced even further, as we are using a rather demanding card with 215 W TDP and this could be replaced by a smaller one such as the Intel PAC with Arria 10 GX FPGA with only 66 W TDP (which could lead to a small performance compromise, but should be sufficient for smaller IBF data structure sizes like the one used in our benchmarks).

6 CONCLUSION

In conclusion, we presented an FPGA design for a state-of-the-art data structure to distribute approximate text queries with direct applications in bioinformatics. Our implementation is up to 5.6 × faster than a highly-tuned CPU implementation and is up to 19 × more energy efficient. The implementation can be seamlessly used in the SeqAn C++ library [20] and is hence available to application programmers in the biomedical domain. We achieved these outcomes by using with OpenCL a high-level language approach for the FPGA design.

ACKNOWLEDGMENTS

We thank Mohamed Issa, Rolf Richter, Graham McKenzie, Hardik Shah, and Klaus-Dieter Oertel from Intel for feedback on various

Table 2: Performance comparison and energy consumption of our work and selected recent FPGA implementations for approximate genomic searches. With the 64 bin FPGA design we achieve the best improvements in performance and energy efficiency of 5.6× and 19×, respectively, over our CPU implementation (numbers in bold). Data and idea taken from [15] and works cited therein. ("SW" means Smith-Waterman algorithm.)

Method & Data Structure	Platform	Device	Reads [Mio]	Length [base pairs]	Time [s]	Speed [Mbp/s]	Energy [J]	Year/Ref
FM-index+SW	Maxeler MAX3	Virtex-6 SX475T	82	90	49.0	151	-	2013/[2]
FM-index	Maxeler MAX3	Virtex-6 SX475T	18	75	13.8	97.8	-	2013/[1]
FM-index	Maxeler MAX5C	Virtex Ultrascale+ VU9P	300	100	683	44.4	-	2021/[15]
IBF, 8192 bins	Intel PAC D5005	Intel Stratix 10 SX	10	100	12.5	79.7	888	this work
			50	100	47.2	105.9	3591	
	Dell PowerEdge T640	Intel Xeon Gold 6248	10	100	36.0	27.8	10 412	
			50	100	161.0	31.1	41 442	
IBF, 64 bins	Intel PAC D5005	Intel Stratix 10 SX	10	100	8.1	122.9	401	this work
			50	100	19.4	**258.4**	**1170**	
	Dell PowerEdge T640	Intel Xeon Gold 6248	10	100	22.5	44.5	5223	
			50	100	108.0	**46.3**	**22 600**	

FPGA related topics. At the Zuse Institute Berlin, this work is partially supported by the German Federal Ministry of Education and Research (BMBF) through grants for the HPCLab within the Research Campus MODAL, project no. 05M20ZBM, and the ORKA-HPC project, grant 01IH17003D. We thank Intel Corp. for providing the Intel PAC with Arria 10 GX FPGA for initial prototype work.

REFERENCES

[1] James Arram, Wayne Luk, and Peiyong Jiang. 2013. Reconfigurable filtered acceleration of short read alignment. In *2013 International Conference on Field-Programmable Technology (FPT)*. 438–441. https://doi.org/10.1109/FPT.2013.6718408

[2] James Arram, Kuen Hung Tsoi, Wayne Luk, and Peiyong Jiang. 2013. Reconfigurable Acceleration of Short Read Mapping. In *2013 IEEE 21st Annual International Symposium on Field-Programmable Custom Computing Machines*. 210–217. https://doi.org/10.1109/FCCM.2013.57

[3] Timo Bingmann, Phelim Bradley, Florian Gauger, and Zamin Iqbal. 2019. COBS: A Compact Bit-Sliced Signature Index BT - String Processing and Information Retrieval. In *String Process. Inf. Retr.* Vol. 11811. Springer, Cham, Cham, 285–303. https://link.springer.com/chapter/10.1007/978-3-030-32686-9{_}21papers3://publication/doi/10.1007/978-3-030-32686-9{_}21

[4] B. H. Bloom. 1970. Space/Time Trade-offs in Hash Coding with Allowable Errors. *Commun. ACM* 13, 7 (July 1970), 422–426. https://doi.org/10.1145/362686.362692

[5] Nae-Chyun Chen, Yu-Cheng Li, and Yi-Chang Lu. 2021. A Memory-Efficient FM-Index Constructor for Next-Generation Sequencing Applications on FPGAs. *CoRR* abs/2102.03045 (2021). arXiv:2102.03045 https://arxiv.org/abs/2102.03045

[6] Rayan Chikhi, Antoine Limasset, and Paul Medvedev. 2016. Compacting de Bruijn graphs from sequencing data quickly and in low memory. *Bioinformatics (Oxford, England)* 32, 12 (June 2016), i201–i208.

[7] Temesgen Hailemariam Dadi, Enrico Siragusa, Vitor C Piro, Andreas Andrusch, Enrico Seiler, Bernhard Y Renard, and Knut Reinert. 2018. DREAM-Yara: an exact read mapper for very large databases with short update time. *Bioinformatics (Oxford, England)* 34, 17 (2018), 766–772.

[8] Fazal Hameed, Asif Ali Khan, and Jeronimo Castrillon. 2021. ALPHA: A Novel Algorithm-Hardware Co-design for Accelerating DNA Seed Location Filtering. *IEEE Transactions on Emerging Topics in Computing* (2021). https://doi.org/10.1109/TETC.2021.3093840

[9] Intel Corp. 2021. *Intel FPGA SDK for OpenCL Pro Edition: Programming Guide*. Intel Corp.

[10] International Human Genome Sequencing Consortium. 2001. Initial sequencing and analysis of the human genome. *Nature* 409, 6822 (2001), 860–921.

[11] Jeremie S. Kim, Damla Senol Cali, Hongyi Xin, Donghyuk Lee, Saugata Ghose, M. Alser, Hasan Hassan, O. Ergin, C. Alkan, and O. Mutlu. 2018. GRIM-Filter: Fast seed location filtering in DNA read mapping using processing-in-memory technologies. *BMC Genomics* 19 (2018).

[12] Daniel Lemire. 2016. *A fast alternative to the modulo reduction*. https://lemire.me/blog/2016/06/27/a-fast-alternative-to-the-modulo-reduction/

[13] Marek Majkowski. 2020. *When Bloom filters don't bloom*. https://blog.cloudflare.com/when-bloom-filters-dont-bloom/

[14] Nathaniel McVicar, Chih Ching Lin, and Scott Hauck. 2017. K-mer counting using bloom filters with an FPGA-attached HMC. *Proc. - IEEE 25th Annu. Int. Symp. Field-Programmable Cust. Comput. Mach. FCCM 2017* (2017), 203–210. https://doi.org/10.1109/FCCM.2017.23

[15] Ho-Cheung Ng, Izaak Coleman, Shuanglong Liu, and Wayne Luk. 2021. Reconfigurable Acceleration of Short Read Mapping with Biological Consideration. In *The 2021 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays* (Virtual Event, USA) *(FPGA '21)*. Association for Computing Machinery, New York, NY, USA, 229–239. https://doi.org/10.1145/3431920.3439280

[16] Ho-Cheung Ng, Shuanglong Liu, Izaak Coleman, Ringo S.W. Chu, Man-Chung Yue, and Wayne Luk. 2020. Acceleration of Short Read Alignment with Runtime Reconfiguration. In *2020 International Conference on Field-Programmable Technology (ICFPT)*. 256–262. https://doi.org/10.1109/ICFPT51103.2020.00044

[17] Ho-Cheung Ng, Shuanglong Liu, and Wayne Luk. 2017. Reconfigurable acceleration of genetic sequence alignment: A survey of two decades of efforts. In *2017 27th International Conference on Field Programmable Logic and Applications (FPL)*. 1–8. https://doi.org/10.23919/FPL.2017.8056838

[18] Prashant Pandey, Fatemeh Almodaresi, Michael A Bender, Michael Ferdman, Rob Johnson, and Rob Patro. 2018. Mantis: A Fast, Small, and Exact Large-Scale Sequence-Search Index. *Cell Syst.* 7, 2 (aug 2018), 201–207.e4. https://linkinghub.elsevier.com/retrieve/pii/S2405471218302394papers3://publication/doi/10.1016/j.cels.2018.05.021https://doi.org/10.1016/j.cels.2018.05.021

[19] Vitor C. Piro, Temesgen H. Dadi, Enrico Seiler, Knut Reinert, and Bernhard Y. Renard. 2020. Ganon: Precise Metagenomics Classification Against Large and Up-To-Date Sets of Reference Sequences. *Bioinformatics* 36, Supplement_1 (jul 2020), I12–I20. https://doi.org/10.1093/BIOINFORMATICS/BTAA458

[20] Knut Reinert, Temesgen Hailemariam Dadi, Marcel Ehrhardt, Hannes Hauswedell, Svenja Mehringer, René Rahn, Jongkyu Kim, Christopher Pockrandt, Jörg Winkler, Enrico Siragusa, Gianvito Urgese, and David Weese. 2017. The SeqAn C++ template library for efficient sequence analysis: A resource for programmers. *Journal of Biotechnology* 261, July (Nov. 2017), 157–168. https://doi.org/10.1016/j.jbiotec.2017.07.017

[21] Tony Robinson, Jim Harkin, and Priyank Shukla. 2021. Hardware acceleration of genomics data analysis: challenges and opportunities. *Bioinformatics* 37, 13 (05 2021), 1785–1795. https://doi.org/10.1093/bioinformatics/btab017 arXiv:https://academic.oup.com/bioinformatics/article-pdf/37/13/1785/39353017/btab017.pdf

[22] Sahand Salamat and T. Simunic. 2020. FPGA Acceleration of Sequence Alignment: A Survey. *ArXiv* abs/2002.02394 (2020).

[23] Enrico Seiler, Svenja Mehringer, Mitra Darvish, Etienne Turc, and Knut Reinert. 2021. Raptor: A fast and space-efficient pre-filter for querying very large collections of nucleotide sequences. *iScience* 24, 7 (July 2021), 102782. https://doi.org/10.1016/j.isci.2021.102782

[24] Y Sireesha and M Roopa. 2015. An FPGA Implementation of Hashed Key-Value Store Using Bloom Filter. *Int. J. Comput. Sci. Mob. Comput.* 4, 5 (2015), 1094–1100.

[25] B. Solomon and C. Kingsford. 2016. Fast search of thousands of short-read sequencing experiments. *Nature Biotechnology* 34, 3 (March 2016), 300–302.

https://doi.org/10.1038/nbt.3442
[26] C. Sun, R. S. Harris, R. Chikhi, and P. Medvedev. 2016. AllSome Sequence Bloom Trees. *bioRxiv* (Dec. 2016), 090464. https://doi.org/10.1101/090464
[27] J. C. Venter, ..., K. Reinert, ..., and X. Zhu. 2001. The sequence of the human genome. *Science* 291 (Feb 2001), 1304–1351.
[28] Derrick E Wood and S Salzberg. 2014. Kraken: ultrafast metagenomic sequence classification using exact alignments. *Genome Biol.* 15, 3 (jan 2014), R46. http://eutils.ncbi.nlm.nih.gov/entrez/eutils/elink.fcgi?dbfrom=pubmed{&}id=24580807{&}retmode=ref{&}cmd=prlinkspapers3://publication/doi/10.1186/gb-2014-15-3-r46papers3://publication/uuid/82DEF96F-5CF8-4C61-B713-63DCB6628C8D