ETH zürich

ETH Library

GateKeeper: a new hardware architecture for accelerating pre-alignment in DNA short read mapping

Journal Article

Author(s):
Alser, Mohammed; Hassan, Hasan; Xin, Hongyi; Ergin, Oğuz; Mutlu, Onur; Alkan, Can

Publication date:
2017-11-01

Permanent link:
https://doi.org/10.3929/ethz-b-000220096

Rights / license:
In Copyright - Non-Commercial Use Permitted

Originally published in:
Bioinformatics 33(21), https://doi.org/10.1093/bioinformatics/btx342

This page was generated automatically upon download from the ETH Zurich Research Collection.
For more information, please consult the Terms of use.

Bioinformatics, 33(21), 2017, 3355–3363
doi: 10.1093/bioinformatics/btx342
Advance Access Publication Date: 31 May 2017
Original Paper

OXFORD

Sequence analysis

GateKeeper: a new hardware architecture for accelerating pre-alignment in DNA short read mapping

Mohammed Alser[1,*], Hasan Hassan[2,3], Hongyi Xin[4], Oğuz Ergin[2], Onur Mutlu[3,*] and Can Alkan[1,*]

[1]Department of Computer Engineering, Bilkent University, Bilkent, Ankara 06800, Turkey, [2]TOBB University of Economics & Technology, Sogutozu, Ankara, Turkey, [3]Department of Computer Science, ETH Zürich, 8092 Zürich, Switzerland and [4]Department of Computer Science, Carnegie Mellon University, Pittsburgh, PA 15213, USA

*To whom correspondence should be addressed.

Associate Editor: Bonnie Berger

Received on November 16, 2016; revised on May 13, 2017; editorial decision on May 24, 2017; accepted on May 29, 2017

Abstract

Motivation: High throughput DNA sequencing (HTS) technologies generate an excessive number of small DNA segments -called *short reads*- that cause significant computational burden. To analyze the entire genome, each of the billions of short reads must be mapped to a reference genome based on the similarity between a read and 'candidate' locations in that reference genome. The similarity measurement, called *alignment*, formulated as an approximate string matching problem, is the computational bottleneck because: (i) it is implemented using quadratic-time dynamic programming algorithms and (ii) the majority of candidate locations in the reference genome do not align with a given read due to high dissimilarity. Calculating the alignment of such incorrect candidate locations consumes *an overwhelming majority* of a modern read mapper's execution time. Therefore, it is crucial to develop a fast and effective filter that can detect incorrect candidate locations and eliminate them before invoking computationally costly alignment algorithms.

Results: We propose GateKeeper, a new hardware accelerator that functions as a *pre-alignment* step that quickly filters out most incorrect candidate locations. GateKeeper is the first design to accelerate pre-alignment using Field-Programmable Gate Arrays (FPGAs), which can perform pre-alignment much faster than software. When implemented on a single FPGA chip, GateKeeper maintains high accuracy (on average >96%) while providing, on average, 90-fold and 130-fold speedup over the state-of-the-art software pre-alignment techniques, Adjacency Filter and Shifted Hamming Distance (SHD), respectively. The addition of GateKeeper as a pre-alignment step can reduce the verification time of the mrFAST mapper by a factor of 10.

Availability and implementation: https://github.com/BilkentCompGen/GateKeeper

Contact: mohammedalser@bilkent.edu.tr or onur.mutlu@inf.ethz.ch or calkan@cs.bilkent.edu.tr

Supplementary information: Supplementary data are available at *Bioinformatics* online.

1 Introduction

High throughput sequencing (HTS) technologies are capable of generating a tremendous amount of sequencing data. For example, the Illumina HiSeq4000 platform can generate more than 1.5 trillion base pairs (bp) in less than four days. This flood of sequenced data continues to overwhelm the processing capacity of existing algorithms and hardware (Canzar and Salzberg, 2015). The success of the medical and genetic applications of HTS technologies relies on

© The Author 2017. Published by Oxford University Press. All rights reserved. For permissions, please e-mail: journals.permissions@oup.com

Downloaded from https://academic.oup.com/bioinformatics/article/33/21/3355/3859176 by ETH Zurich user on 21 October 2021

the existence of sufficient computational resources, which can quickly analyze the overwhelming amounts of data that the sequencers generate. An HTS instrument produces short reads (typically 75–150 bp) sampled randomly from DNA. In the presence of a reference genome, the short reads are first mapped to the long reference sequence. During this process, called *read mapping*, each short read is mapped onto one or more possible locations in the reference genome based on the similarity between the short read and the reference sequence segment at that location. Optimal *alignment* of the read and the reference segment could be calculated using the Smith-Waterman local alignment algorithm (Smith and Waterman, 1981). However, this approach is infeasible as it requires O(mn) running time, where m is the read length (100–150 bp for Illumina) and n is the reference length (~3.2 billion bp for human genome), for each *read* in the dataset (hundreds of millions to billions). Therefore, read mapping algorithms apply heuristics to first find candidate map locations (*seed locations*) of subsequences of the reads using hash tables (Alkan *et al.*, 2009; David *et al.*, 2011; Hach *et al.*, 2010; Homer *et al.*, 2009; Xin *et al.*, 2013) or BWT-FM indices (Langmead and Salzberg, 2012; Langmead *et al.*, 2009; Li and Durbin, 2009; Li *et al.*, 2004), and then align the read in full *only* to those seed locations. Although the strategies for finding seed locations vary among different read mapping algorithms, seed location identification is typically followed by a *verification* step, which compares the read to the reference segment at the seed location to check if the read aligns to that location in the genome with fewer differences than a threshold. The verification step is the dominant part of the whole execution time in current mappers (over 90% of the running time) (Cheng *et al.*, 2015; Xin *et al.*, 2013). It calculates *edit distance* using quadratic-time algorithms such as Levenshtein's edit distance (Levenshtein, 1966), Smith-Waterman (Smith and Waterman, 1981) and Needleman-Wunsch (Needleman and Wunsch, 1970). Edit distance is defined as the minimum number of edits (i.e. insertions, deletions, or substitutions) needed to make the read exactly match the reference segment (Levenshtein, 1966). If the edit distance score is greater than a user-defined *edit distance threshold* (usually less than 5% of the read length (Ahmadi *et al.*, 2012; Hatem *et al.*, 2013; Xin *et al.*, 2015)), then the mapping is considered to be invalid (i.e. the read does not match the segment at seed location) and thus is rejected.

Definition 1. *Given a candidate read r, a reference segment f, and an edit distance threshold E, the pairwise alignment problem is to identify a set of matches of r in f, where the read aligns with an edit distance* $\leq E$.

Recent work found that an overwhelming majority (>98%) of the seed locations exhibit more edits than the threshold (Xin *et al.*, 2013, 2015). These particular seed locations impose a large computational burden as they waste 90% of the mapper's execution time in verifying these incorrect mappings (Cheng *et al.*, 2015; Xin *et al.*, 2013). To tackle these challenges and bridge the widening gap between the execution time of the mappers and the increasing amount of sequencing data, most existing works fall into two approaches: (i) Design hardware accelerators to *accelerate the verification step* (Arram *et al.*, 2013; Houtgast *et al.*, 2015; Liu *et al.*, 2012; Luo *et al.*, 2013; Olson *et al.*, 2012; Waidyasooriya *et al.*, 2014). (ii) Build software-based *alignment filters* before the verification step (Cheng *et al.*, 2015; Marco-Sola *et al.*, 2012; Rasmussen *et al.*, 2006; Ukkonen, 1992; Weese *et al.*, 2009, 2012; Xin *et al.*, 2013, 2015). Such filters aim to minimize the number of candidate locations on which alignment is performed. They calculate a best guess estimate for the alignment score between a read and a seed location on the reference. If the lower bound exceeds a certain number of edits, indicating that the read and the segment at the seed location do not align, the seed location is eliminated such that no alignment is performed. Unfortunately, existing filtering techniques are either slow, such as Shifted Hamming distance (SHD) (Xin *et al.*, 2015), or inaccurate in filtering, such as the Adjacency Filter (Xin *et al.*, 2013) (implemented as part of FastHASH (Xin *et al.*, 2013)) and mrsFAST-Ultra (Hach *et al.*, 2014)). While mrsFAST-Ultra is able to detect only substitutions, FastHASH is unable to tolerate substitutions efficiently. We provide full descriptions of the key principles underlying each strategy in Supplementary Material, Section S1.2.

Our goal, in this work, is to minimize the mapper time spent on accurate alignment filtering. To this end, we introduce a new FPGA-based fast alignment filtering technique (called GateKeeper) that acts as a pre-alignment step in read mapping. To our knowledge, this is the first work that provides a new pre-alignment algorithm and architecture using reconfigurable hardware platforms. A fast filter designed on a specialized hardware platform can drastically expedite alignment by *reducing the number of locations that must be verified via dynamic programming*. This eliminates many unnecessary expensive computations, thereby greatly improving overall run time.

Our filtering technique improves and accelerates the state-of-the-art SHD filtering algorithm (Xin *et al.*, 2015) using new mechanisms and FPGAs. We build upon the SHD algorithm as it is the fastest and the most accurate filter (Xin *et al.*, 2015). Our new filtering algorithm has two properties that make it suitable for an FPGA-based implementation: (i) it is highly parallel, (ii) it heavily relies on bitwise operations such as shift, XOR and AND. Due to the highly parallel and bitwise-processing-friendly architecture of modern FPGAs, our design achieves more than two orders of magnitude speedup compared to the best prior software-based filtering approaches (SHD and Adjacency Filter), as our comprehensive evaluation shows (Section 3). Our architecture discards the incorrect mappings from the candidate mapping pool in a streaming fashion – data is processed as it is transferred from the host system. Filtering the mappings in a streaming fashion gives the ability to integrate our filter with any mapper that performs alignment, such as Bowtie2 (Langmead and Salzberg, 2012) and BWA-MEM (Li, 2013).

Contributions. We make the following contributions:

- We introduce the **first** hardware acceleration system for alignment filtering, called GateKeeper, which greatly reduces the need for alignment verification in DNA read mapping. To this end, we develop both a hardware-acceleration-friendly filtering algorithm and a highly parallel hardware accelerator design. We show that developing a hardware-based alignment filtering algorithm and architecture together is both feasible and effective by building our accelerator on a modern FPGA system.
- We comprehensively evaluate GateKeeper and compare it to two state-of-the-art software-based alignment filtering algorithms. A key result is that our design for reads of length 100 bp on a single FPGA chip provides, on average, 90-fold and 130-fold speedup over the state-of-the-art filters, Adjacency Filter (Xin *et al.*, 2013) and SHD (Xin *et al.*, 2015), respectively. Experimental results on both simulated and real datasets demonstrate that GateKeeper has a low false positive rate (the rate of incorrect mappings that are accepted by the filter) of 4% on average.
- We provide the design and implementation of a complete FPGA system and release its source code. To our knowledge, GateKeeper is the first open-source, freely available FPGA based alignment filter for genome analysis.

Downloaded from https://academic.oup.com/bioinformatics/article/33/21/3355/3859176 by ETH Zurich user on 21 October 2021

2 Gatekeeper architecture

2.1 Overview of our accelerator architecture

Based on the discussion provided in the Supplementary Material, Section 1.2, we introduce the **first** specialized FPGA-friendly hardware architecture for a *new alignment filtering algorithm*. The overall architecture, implementation details and flowchart representation of GateKeeper are discussed in the Supplementary Material, Section 1.3.1. Our current filter implementation relies on several optimization methods to create a robust and efficient filtering approach. At both the design and implementation stages, we satisfy several requirements: (i) Ensuring a lossless filtering algorithm by preserving all correct mappings. (ii) Supporting both Hamming distance and edit distance. The Hamming distance is a special case of the edit-distance. It is defined as the minimum number of substitutions required to change the read into the reference segment. The Hamming distance is computed in linear time. (iii) Examining the alignment between a read and a reference segment in a fast and efficient way (in terms of execution time and required resources).

2.2 Parallelization

GateKeeper is designed to utilize the large amounts of parallelism offered by FPGA architectures (Aluru and Jammula, 2014; Herbordt *et al.*, 2007; Trimberger, 2015). The use of FPGAs can yield significant performance improvements, especially for massively parallel algorithms. FPGAs are the most commonly used form of reconfigurable hardware engines today, and their computational capabilities are greatly increasing every generation due to increased number of transistors on the FPGA chip. An FPGA chip can be programmed (i.e. configured) to include a very large number of hardware execution units that are custom-tailored to the problem at hand. We take advantage of the fact that alignment filtering of one read is *inherently independent* of filtering of another read. We therefore can examine many reads in a parallel fashion. In particular, instead of handling each read in a sequential manner, as CPU-based filters (e.g. SHD) do, we can process a large number of reads at the same time by integrating as many hardware filtering processing cores as possible (constrained by chip area) in the FPGA chip. Each processing core is a complete alignment filter and can handle a single read at a time. We use the term 'processing core' in this paper to refer to the entire operation of the filtering process involved in GateKeeper. Processing cores are part of our architecture and are unrelated to the term 'CPU cores' or 'threads'.

2.3 GateKeeper processing core

Our primary purpose is to enhance the state-of-the-art SHD alignment filter such that we can greatly accelerate pre-alignment by taking advantage of the capabilities and parallelism of FPGAs. To achieve our goal, we design an algorithm inspired by SHD to reduce both the utilized resources and the execution time. These optimizations enable us to integrate more processing cores within the FPGA chip and hence examine many alignments at the same time. We present three new methods that we use in each GateKeeper processing core to improve execution time. Our first method introduces a new algorithmic method for performing alignment very rapidly compared to the original SHD. This method provides: (1) fast detection for exact matching alignment and (2) handling of one or more base-substitutions. Our second method supports calculating the edit distance with a new, very efficient hardware design. Our third method addresses the problem of hardware resource overheads introduced due to the use of FPGA as an acceleration platform. We provide the workflow of GateKeeper including the three optimization methods in the Supplementary Material, Figure S8. All features are implemented within the filtering processing core hardware and thus are performed highly efficiently. Next, we describe the three new methods.

2.3.1 Method 1: Fast approximate string matching

We first discuss how to examine the alignment of reads against the reference sequence with a given Hamming distance threshold, and later extend our solution to support edit distance. Our first method aims to quickly detect the obviously-correct alignments that contain no edits or only few substitutions (i.e. less than the user-defined threshold). If the first method detects a correct alignment, then we can skip the other two methods but we still need the optimal alignment algorithms. A read is mappable if the Hamming distance between the read and its seed location does not exceed the given Hamming distance threshold. Hence, the first step is to identify all bp matches by calculating what we call a Hamming mask. The Hamming mask is a bit-vector of '0's and '1's representing the comparison of the read and the reference, where a '0' represents a bp match and a '1' represents a bp mismatch. We need to count only occurrences of '1' in the Hamming mask and examine whether their total number is equal to or less than the user-defined Hamming distance threshold. If so, the mapping is considered to be valid and the read passes the filter. Similarly, if the total number of '1' is greater than the Hamming distance threshold then we cannot be certain whether this is because of the high number of substitutions, or there exist insertions and/or deletions; hence, we need to follow the rest of our algorithm. Our filter can detect not only substitutions but also insertions and deletions in an efficient way, as we discuss next.

2.3.2 Method 2: Insertion and deletion (indel) detection

Our indel detection algorithm is inspired by the original SHD algorithm presented in (Xin *et al.*, 2015). If the substitution detection rejects an alignment, then GateKeeper checks if an insertion or deletion causes the violation (i.e. high number of edits). Figure 1 illustrates the effect of occurrence of edits on the alignment process. If there are one or more base-substitutions or the alignment is exact matching, the matching and mismatching regions can be accurately determined using Hamming distance. As the substitutions have no effect on the alignment of subsequent bases, the number of edits is equivalent to the number of '1's in the resulting Hamming mask. On the other hand, each insertion and deletion can shift multiple trailing bases and create multiple edits in the Hamming mask. Thus, pairwise comparison (bitwise XOR) between the bases of the read and the reference segment is not sufficient. Our indel detection method identifies whether the alignment locations of a read are valid, by shifting individual bases. We need to perform E incremental shifts to the right direction to detect any read that has E deletions, where E is the edit distance threshold. The right shift process guarantees to cancel the effect of deletion. Similarly, we need to perform E incremental shifts to the left direction to detect any read that has E insertions. As we do not have prior knowledge about whether there exist insertions, or deletions, or both, we need to test for every possible case in our algorithm. Thus, GateKeeper generates $2E$ Hamming masks regardless the source of the edit. Each mask is generated after incrementally shifting the candidate read against the reference and performing pairwise comparison (i.e. bitwise XOR operation). A segment of consecutive matches in the one-step right-shifted mask

Downloaded from https://academic.oup.com/bioinformatics/article/33/21/3355/3859176 by ETH Zurich user on 21 October 2021

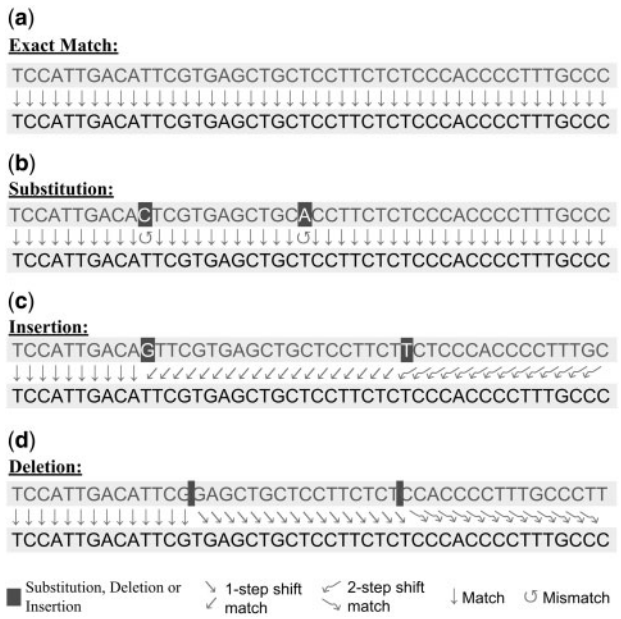

Fig. 1. An example showing how various types of edits affect the alignment of two reads. In (**a**) the upper read exactly matches the lower read and thus each base exactly matches the corresponding base in the target read. (**b**) shows base-substitutions that only affect the alignment at their positions. (**c**) and (**d**) demonstrate insertions and deletions, respectively. Each edit has an influence on the alignment of all the subsequent bases

indicates that there is a single deletion that occurred in the read sequence.

Since deletions and insertions affect only the trailing bases, we need to have an additional Hamming mask that is generated with no shifts. This mask helps detect the matches that are located before the first indel. However, this mask is already generated as part of the first method of the algorithm (i.e. Fast Approximate String Matching). The last step is to merge all the $2E+1$ Hamming masks using a bitwise AND operation. This step tells us where the relevant matching and mismatching regions reside in the presence of edits in the read compared to the reference segment. We provide an example of a candidate alignment with all masks that are generated by a single GateKeeper processing core in the Supplementary Material, Figure S9. Identical regions are identified in each shifted Hamming mask as streaks of continuous '0's. As we use a bitwise AND operation, a zero at any position in the $2E+1$ Hamming masks leads to a '0' in the resulting final bit-vector at the same position. Hence, even if some Hamming masks show a mismatch at that position, a zero in some other masks leads to a match ('0') at the same position. This tends to underestimate the actual number of edits and eventually causes some incorrect mappings to pass. To fix this issue, we build a *new hardware-based amending process*. The amending process is first presented in the original SHD filter (Xin *et al.*, 2015) that actually amends (or *flips*) short streaks of '0's (single or double zeros) in each mask into '1's such that they do not mask out '1's in other Hamming masks. Short streaks of '0's do not represent identical sections and thus they are useless. As a result, bit streams such as 101, 1001 are replaced with 111 and 1111, respectively. In SHD, the amending process is accomplished using a 4-bit packed shuffle (SIMD parallel table-lookup instruction), shift and OR operations. The number of computations needed is 4 packed shuffle, $4m$ bitwise

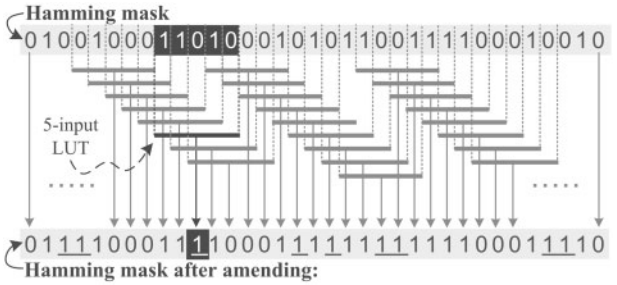

Fig. 2. Workflow of the proposed architecture for the parallel amendment operations

OR, and three shift operations for each Hamming mask, which is $(7+4m)(2E+1)$ operations, where m is the read length. We find that this is very inefficient for FPGA implementation. To reduce the number of operations, we propose using dedicated hardware components in FPGA slices. More precisely, rather than shifting the read and then performing packed shuffle to replace patterns of 101 or 1001 to 111 or 1111 respectively, we perform only packed shuffle *independently and concurrently* for each bit of each Hamming mask. As illustrated in Figure 2, the proposed architecture for amendment operations contains one 5-input look-up table (LUT) dedicated for each output bit, except the first and last output bits. We provide full details of our amending architecture in the Supplementary Material (Section 1.3). Using this dedicated architecture, we are able to get rid of the four shifting operations and perform the amending process concurrently for all bits of any Hamming mask. Thus, *the required number of operations is only $(2E+1)$ instead of $(7+4m)(2E+1)$ for a total of $(2E+1)$ Hamming masks*. This saves a considerable amount of the filtering time, reducing it by 407× for a read that is 100 bp long.

2.3.3 Method 3: Minimizing hardware resource overheads

The short reads are composed of a string of nucleotides from the DNA alphabet $\sum=\{A, C, G, T\}$. Since the reads are processed in an FPGA platform, the symbols have to be encoded in to a unique binary representation. We need 2 bits ($\log_2|\sum|$ bits) to encode each symbol. Hence encoding a read sequence of length m results in a $2m$-bit word. Encoding the reads into a binary representation introduces overhead to accommodate not only the encoded reads but also the Hamming masks as their lengths also double (i.e. $2m$). The issue introduced by encoding the read can be even worse when we apply certain operations on these Hamming masks. For example, the number of LUTs required for performing the amending process on the Hamming masks will be doubled, mainly due to encoding the read. To reduce the complexity of the subsequent operations on the Hamming masks and save about half of the required amount of FPGA resources, we propose a new solution. We observe that comparing a pair of DNA nucleotides is similar to comparing their binary representations (e.g., comparing A to T is similar to comparing '00' to '11'). Hence, comparing each two bits from the binary representation of the read with their corresponding bits of the reference segment generates a single bit that represents one of two meanings; either match or mismatch between two bases. This is performed by encoding each two bits of the result of the pairwise comparison (i.e. bitwise XOR) into a single bit of '0' or '1' using OR operations in a

Downloaded from https://academic.oup.com/bioinformatics/article/33/21/3355/3859176 by ETH Zurich user on 21 October 2021

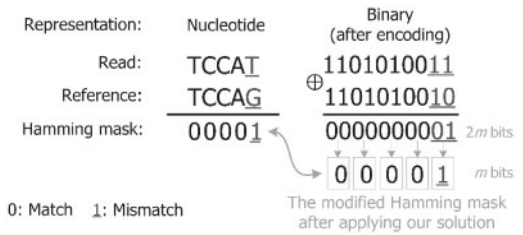

Fig. 3. An example of applying our solution for reducing the number of bits of each Hamming mask by half. We use a modified Hamming mask to store the result of applying the bitwise OR operation to each two bits of the Hamming mask. The modified mask maintains the same meaning of the original Hamming mask

Table 1. Overall benefits of GateKeeper over SHD in terms of number of operations performed

# of operations for SHD:	
– $m(2E+1)$ bitwise XOR[b].	– $4m(2E+1)$ bitwise OR.[a]
– $2E$ shift.	– $4(2E+1)$ packed shuffle.[a]
– $3(2E+1)$ shift.[a]	
# of operations for GateKeeper:	
For Substitution Detection	For Indel Detection
– $2m$ bitwise XOR.	– $2m(2E+1)$ bitwise XOR.
	– $2E$ shift.
	– $m(2E+1)$ bitwise OR.
	– $(2E+1)$ look-up table.[a]

[a]This operation is required for the amending process.
[b]E: edit distance threshold. m: read length.

parallel fashion, as explained in Figure 3. This makes the length of each Hamming mask equivalent to the length of the original read, without affecting the meaning of each bit of the mask. The modified Hamming masks are then merged together in $2E$ bitwise AND operations. Finally, we count the number of ones (i.e. edits) in the final bit-vector mask; if the count is less than the edit distance threshold, the filter accepts the mapping.

2.4 Novelty

GateKeeper is the only read mapping filter that takes advantage of the parallelism offered by FPGA architectures in order to expedite the alignment filtering process. GateKeeper supports both Hamming distance and edit distance in a fast and efficient way. Each GateKeeper processing core performs all operations defined in the GateKeeper algorithm (Supplementary Material, Section 1.3, Algorithm 1). Table 1 summarizes the relative benefits gained by each of the aforementioned optimization methods over the best previous filter, SHD (E is the user-defined edit distance threshold and m is the read length). When a read matches the reference exactly, or with few substitutions, GateKeeper requires *only 2m bitwise XOR operations*, providing substantial speedup compared to SHD, which performs a much greater number of operations. However, this is not the only benefit we gain from our first proposed method (i.e. Fast Approximate String Matching). As this method provides an accurate examination for alignments with only substitutions (i.e. no deletions or insertions), we can directly skip calculating their optimal alignment using the computationally expensive alignment algorithms (i.e. verification step). For more general cases such as deletions and insertions, GateKeeper still requires far fewer operations (as shown in Table 1) than the original SHD filter, due to the optimization methods outlined above. Our improvements over SHD help drastically reduce the execution time of the filtering process. The rejected alignments by our GateKeeper filter are *not* further examined by a verification step. Thus, GateKeeper leads to the acceleration of the entire read mapping process, as our evaluation quantitatively shows (Section 3).

Table 2. FPGA resource utilization for a single GateKeeper core

	Resource utilization %				
Read length	100 bp		300 bp		
Edit distance	2	5	2	5	15
Slice LUT[a]	0.39%	0.71%	1.27%	2.2%	4.82%
Slice Register[b]	0.01%	0.01%	0.01%	0.01%	0.01%

[a]LUT: look-up tables.
[b]Flip-flop.

3 Evaluation

To implement and evaluate GateKeeper, we use a Xilinx VC709 board (Xilinx, 2014), which features a Virtex-7 XC7VX690T-2FFG1761C FPGA (Xilinx, 2015), and a 3.6 GHz Intel i7-3820 CPU with 8 GB RAM as the host and to run all experiments. We build the FPGA design with Vivado 2014.4 in Verilog. We use RIFFA 2.2 (Jacobsen *et al.*, 2015) to perform the host-FPGA PCIe communication. We configure RIFFA 2.2 as Gen3 4-lane PCIe.

3.1 Theoretical speedup

We first examine the maximum speedup theoretically possible with our architecture, assuming the only constraint in the system is the FPGA logic. To this end, we calculate the number of mappings that our accelerator board can potentially examine in parallel using as many GateKeeper processing cores as possible. Table 2 shows the resource utilization of a single processing core for two read lengths of 100 and 300 bp, with different edit distance thresholds. We find that a single processing core for a read length of 300 bp shows 3-fold increase in the number of LUTs compared to its counterpart for a read length of 100 bp, for the same edit distance threshold. This observation is supported by theory: as we show in Table 1, the number of operations of GateKeeper is proportional to both read length and edit distance threshold. Based on the resource report in Table 2, we estimate that we can design GateKeeper, on the VC709 FPGA, to process up to 140 alignments of 100 bp reads and edit distance threshold of up to 5% in parallel in a single clock cycle. The number of alignments drops to 20 for a read length of 300 bp and $E=15$. The bottleneck in this idealized system is transferring a total of 28 000 (140 alignment $\times$ 100 bp $\times$ 2 bits for encoding) bits in a single clock cycle into the FPGA, which is not practical for any of the existing PCIe drivers that supply data to the FPGA. For instance, RIFFA (Jacobsen *et al.*, 2015) transmits the mapping pairs into the FPGA in 'packages' of 128 bits per clock cycle at a clock speed of 250 MHz (i.e. 4 nanoseconds). We conclude that the theoretical speedup provided by GateKeeper is extremely large, but practical speedup, which we will examine next, is mainly limited by the data transfer rate into the accelerator.

Downloaded from https://academic.oup.com/bioinformatics/article/33/21/3355/3859176 by ETH Zurich user on 21 October 2021

Table 3. Overall system resource utilization under different read lengths and edit distance thresholds

	Resource utilization %			
Read length	100 bp		300 bp	
Edit distance	2	5	2	15
Slice LUT	32%	45%	50%	69%
Slice register	2%	2%	17%	91%
Block memory	2%	2%	2%	2%

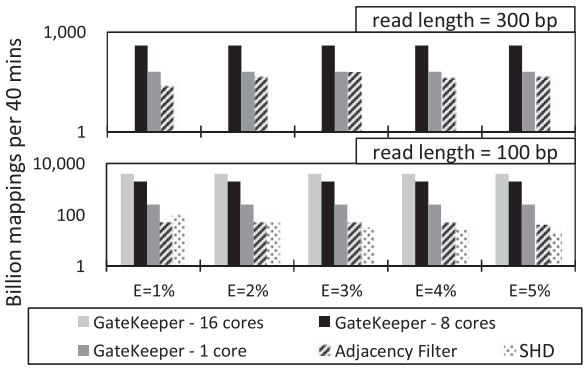

Fig. 4. Performance of GateKeeper, SHD, and the Adjacency Filter in terms of the number of examined mappings across different edit distance thresholds and read lengths. The y-axis is on a logarithmic scale. SHD does not support 300 bp long reads

3.2 Experimental speedup

Throughput and resource analysis. Filtering speed of GateKeeper is dependent on the total number of concurrent processing cores and the clock frequency. The number of processing cores is determined by the maximum data throughput and the available FPGA resources. The operating frequency of the accelerator is 250 MHz. At this frequency, we observe a data throughput of nearly 3.3 GB/s, which corresponds to ∼13.3 billion bases per second, nearly reaching the maximum throughput of 3.64 GB/s provided by the RIFFA communication channel that feeds data into the FPGA (Jacobsen *et al.*, 2015). Table 3 lists the resource utilization of the entire design including the PCIe communication logic, for various read lengths and edit distance thresholds. For a read length of 100 bp, we find that we can align each read against up to 16 different reference segments in parallel, without violating the timing constraints (e.g. maximum operating frequency). This design occupies about 50% of the available FPGA resources (i.e. slice LUTs). We find that as read length increases, timing constraints of the design can be violated. By pipelining the design (i.e. shortening the critical path delay of each processing core by dividing it into stages or smaller tasks), we can meet the timing constraints and achieve more parallelism. However, pipelining the design comes with the expense of increased register utilization. For a read length of 300 bp, GateKeeper can process up to 8 alignments concurrently and use 91% of the available registers. As our design is FPGA-platform independent, FPGAs with higher logic density (such as Xilinx UltraScale+ FPGAs) can be used, to achieve more parallelism and higher data throughput. Next, we evaluate the effect of varying the number of processing cores on the execution time of GateKeeper.

Speedup versus existing filters. We now evaluate the execution time of GateKeeper compared to the best existing filters. We use mrFAST (Alkan *et al.*, 2009) mapper to retrieve all potential mappings (read-reference pairs) from two datasets. The first set (ERR240727_1) contains about 4 million real reads, each of length 100 bp, from the 1000 Genomes Project Phase I (Consortium, 2012). The second set contains about 100 thousand reads, each of length 300 bp, simulated from the human genome using the *mason* simulator (http://packages.seqan.de/mason/). Figure 4 shows the number of mappings that are processed by GateKeeper (with different numbers of processing cores), SHD, and the Adjacency Filter within 40 minutes. To ensure as fair a comparison as possible, we evaluate Gate Keeper using a single FPGA chip and run both SHD and the Adjacency Filter using a single CPU core. We believe our comparison is fair because we compare GateKeeper running on a part of a single FPGA chip to SHD/Adjacency-Filter running on a part of a single CPU (Section 1.5, Supplementary Material). Both SHD and the Adjacency Filter are software filters (i.e. cannot run on an FPGA) and they do not support multithreading. SHD supports a read length up to only 128 bp (due to SIMD registers size). Under different edit distance thresholds (up to 5% of the read length), GateKeeper provides consistently good performance.

On average, GateKeeper for 100 bp reads is 130x faster than SHD and 90× faster than the Adjacency Filter. For longer reads (i.e. 300 bp), GateKeeper is also, on average, 10× faster than the Adjacency Filter. As edit distance threshold increases, Gatekeeper's speedup over SHD and the Adjacency Filter also increases (e.g. up to 105× and 215× faster than the Adjacency Filter and SHD, respectively, when $E = 5$ edits and read length = 100 bp). This is because our architecture offers the ability to perform all computations in a parallel fashion (as we explained when we described our three new methods in the GateKeeper core). Note that the Adjacency Filter becomes faster than SHD as E increases, but at the expense of accuracy, as we will show soon. We conclude that GateKeeper greatly improves the performance of alignment filtering by at least one order of magnitude. GateKeeper also scales very well over a wide range of both edit distance thresholds and read lengths.

3.3 Filtering accuracy

An ideal filter should be both fast and accurate in rejecting the incorrect mappings. We evaluate the accuracy of GateKeeper by computing its true negative, false positive and false negative rates. We use the Needleman-Wunsch algorithm to benchmark the three filters as this algorithm has both zero false positive and zero false negative rates. To evaluate the accuracy of SHD regardless of the limitation of its SIMD implementation (i.e. limited read length), we implement SHD in C and refer to it as SHD-C. We also compare the accuracy of our filter with SHD and the Adjacency Filter using both simulated and real mapping pairs. We simulate reads from the human genome using the *mason* simulator. The configuration and parameters used in our experiment are provided in Supplementary Material (Section 1.4). We generate five sets, each of which contains 400 000 Illumina-like reads. Each set has an equal number of reads of length 64, 100, 150 and 300 bp. While two sets have a low number of different types of edits, the other three sets have a high number of substitutions, insertions and deletions. The purpose of simulating the low-edit reads is that we want most of the reads to have edits less than the allowed threshold. This enables us to quantify the false negatives (i.e. correct mappings that are rejected by the filter) of the three filters with different read lengths. On the other hand, we use the edit-rich reads to evaluate the robustness of the three filters to

Downloaded from https://academic.oup.com/bioinformatics/article/33/21/3355/3859176 by ETH Zurich user on 21 October 2021

incorrect mappings. This enables us to quantify both the false positives and true negatives. While the false positive rate is the rate of incorrect mappings that are accepted by the filter, the true negative rate is the rate of incorrect mappings that are rejected by the filter. Figure 5(a) shows the result of this experiment. We also consider a more realistic scenario in which reads can have a combination of substitutions and indels. Instead of simulated reads, we use the first 30 million pairs produced by mrFAST when the dataset ERR240727_1 mapped to the human genome to evaluate both the false positive and true negative rates of the three filters, as shown in Figure 5(b).

Based on these results, we make five main observations. (i) Using the low-edit reads, we observe that the three filters *never filter out correct mappings*; hence, they provide a lossless filtering mechanism with a false negative rate of zero. (ii) We find that GateKeeper is very effective and superior to the Adjacency Filter at both substitution and indel detection. Figure 5(a) shows the average false positive and true negative rates of the three filters, respectively, using the three simulated edit-rich sets. We observe that both GateKeeper and SHD have the same false positive and true negative rates. (iii) On average, GateKeeper produces a false positive rate of 4%, which is much smaller (on average, 0.25×) than that of the Adjacency Filter. (iv) GateKeeper rejects a significant fraction of incorrect mappings (e.g. 84% to 99.9% of the mappings, depending on the edit distance threshold used) and thus avoids expensive verification computations required by alignment algorithms. GateKeeper rejects up to 20% more incorrect mappings than the Adjacency Filter. (v) The Adjacency Filter is more robust in handling indels than in handling substitutions. This is expected as the presence of one or more substitutions in any seed is counted by the Adjacency Filter as a single mismatch. The effectiveness of the Adjacency Filter for substitutions and indels diminishes when E becomes larger than 3%. The detailed results for each of the three edit-rich sets are provided in the Supplementary Material (Section 1.4). We conclude that Gatekeeper's accuracy is as good as that of the best previous filter, SHD, and much better than that of the Adjacency Filter yet GateKeeper is much faster than both SHD and the Adjacency Filter (as we showed earlier). Hence, GateKeeper is extremely fast and accurate.

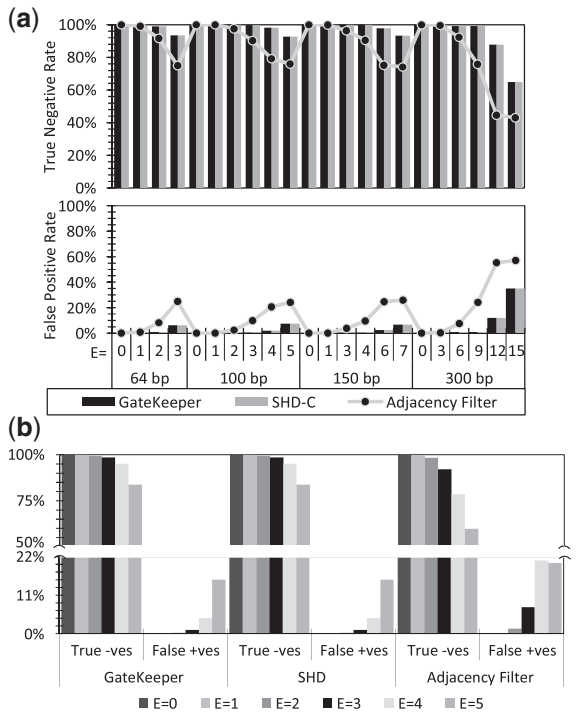

Fig. 5. Accuracy of GateKeeper, SHD and the Adjacency Filter across different edit distance thresholds (E) and read lengths. We calculate the false positive [False +ves] and true negative [True -ves] rates using (**a**) simulated and (**b**) real mapping pairs

3.4 Verification

GateKeeper is a standalone filter and can be integrated with any existing reference-based mapper. GateKeeper does *not* replace the local/global alignment algorithms (e.g. Smith–Waterman (Smith and Waterman, 1981) and Needleman–Wunsch (Needleman and Wunsch, 1970)). GateKeeper should be followed by an alignment verification step, which precisely verifies the alignments that pass our filter and eliminates the false positives (as provided in the Supplementary Material, Fig. S9). The verification step is accurate and admits zero false positive rate. It also allows specifying a cost to each edit (i.e. a scoring system). Such integration is mapper-specific and will be explored in detail for various mappers in our future research. In this work, we mainly focus on and deeply evaluate the benefits and downsides of our filtering algorithm and architecture independently of any mapper it can be combined with. Nonetheless, we have a preliminary assessment on the overall benefits of integrating GateKeeper with the mrFAST mapper (Alkan *et al.*, 2009). We select mrFAST for two main reasons. (i) It already includes the Adjacency Filter (Xin *et al.*, 2013) as a pre-alignment step, so it constitutes a state-of-the-art baseline. (ii) It utilizes a banded Levenshtein edit distance algorithm (Ukkonen, 1985) that is parallelized using the Intel SSE instructions, and thus it utilizes the capabilities of state-of-the-art hardware. Table 4 summarizes the effect of pre-alignment on the overall mapping time, when all reads from ERR240727_1 (100 bp) and Set_5 (300 bp, mason-simulated deletion-rich reads) are mapped to the human genome with an edit distance threshold of 5%. We make three observations. (i) GateKeeper is at least 41 times faster than the banded dynamic programming alignment algorithm (Ukkonen, 1985). (ii) The verification time drops by a factor of 10 after replacing the Adjacency Filter with GateKeeper as the pre-alignment step. (iii) GateKeeper reduces

Table 4. Overall mrFAST mapping time (in hours) with and without a pre-alignment step, with an edit distance threshold of 5%

Read length/E	mrFAST version/pre-alignment type	Filtering & verification time (speed-up)	Overall mapping time (speed-up)
100 bp /5 edits	2.1/No Pre-alignment	22.60 h (1×)	24.27 h (1×)
	2.6/Adjacency Filter	5.65 h (4×)	7.31 h (3.3×)
	2.1/GateKeeper	0.55 h (41×)	2.50 h (9.7×)
300 bp /15 edits	2.1/No Pre-alignment	0.94 h (1×)	1.02 h (1×)
	2.6/Adjacency Filter	0.04 h (24×)	0.12 h (8×)
	2.1/GateKeeper	0.003 h (279×)	0.09 h (11×)

Downloaded from https://academic.oup.com/bioinformatics/article/33/21/3355/3859176 by ETH Zurich user on 21 October 2021

the overall mapping time of mrFAST (mrFAST-2.6) by a factor of 1.3–3. Details are provided in the Supplementary Material, Section 1.6.

4 Future work

GateKeeper shows that there is a great benefit in designing an alignment filtering accelerator to handle the flood of sequenced data. Since a single-core GateKeeper has only a small footprint on the FPGA, we can combine our architecture with any of the FPGA-based accelerators for BWT-FM or hash-based mapping techniques on a single FPGA chip. With such a combination, the end result would be an efficient and fast multi-layer mapping system: alignments that pass GateKeeper can be further verified using a dynamic programing based alignment algorithm *within* the same chip. We leave this combination for future work. Another potential target of our research is to influence the design of more intelligent and attractive sequencing machines by integrating GateKeeper inside them, to perform real-time pre-alignment. This approach has two benefits. First, it can hide the complexity and details of the underlying hardware from users who are not necessarily fluent in FPGAs (e.g. biologists and mathematicians). Second, it allows a significant reduction in total genome analysis time by starting read mapping while still sequencing (Lindner *et al.*, 2016). Our next efforts will also focus on investigating the sources of the false positives and explore the possibility of eliminating them to achieve a dynamic-programming-free alignment approach or a more accurate filter.

5 Summary

In this paper, we propose the first hardware accelerator architecture for pre-alignment in genome read mapping. In our experiments, GateKeeper can filter up to ~4 trillion mappings within 40 mins using a single FPGA chip while preserving all correct ones. Comparison against the best two software-based alignment filters reveals the following: (i) Our filter provides, on average, 90-fold and 130-fold speedup compared to the Adjacency Filter and SHD, respectively. (ii) Our filter is as accurate as the SHD and 4 times more accurate than the Adjacency Filter. We conclude that GateKeeper is both a fast and an accurate filter that can improve the performance of existing and future read mappers. Our preliminary results show that the addition of GateKeeper as the pre-alignment step can reduce the filtering and verification time of the mrFAST mapper by a factor of 10.

Our design is open source and freely available online. To our knowledge, GateKeeper is the first open-source FPGA-based alignment filtering accelerator for genome analysis. As such, we hope that it catalyzes the development and adoption of such hardware accelerators in genome sequence analysis, which are becoming increasingly necessary to cope with the processing requirements of greatly increasing amounts of genomic data.

Funding

This study is supported by NIH Grant (R01 HG006004 to O. Mutlu and C. Alkan) and a Marie Curie Career Integration Grant (PCIG-2011-303772) to C. Alkan under the Seventh Framework Programme. M. Alser also acknowledges support from the Scientific and Technological Research Council of Turkey, under the TUBITAK 2215 program.

Conflict of Interest: none declared.

References

Ahmadi,A. *et al.* (2012) Hobbes: optimized gram-based methods for efficient read alignment. *Nucleic Acids Res.*, **40**, e41–e41.

Alkan,C. *et al.* (2009) Personalized copy number and segmental duplication maps using next-generation sequencing. *Nature Genet.*, **41**, 1061–1067.

Aluru,S. and Jammula,N. (2014) A review of hardware acceleration for computational genomics. *Des. Test IEEE*, **31**, 19–30.

Arram,J. *et al.* (2013) Reconfigurable acceleration of short read mapping. In: *Field-Programmable Custom Computing Machines (FCCM), 2013 IEEE 21st Annual International Symposium on IEEE*, pp. 210–217.

Canzar,S. and Salzberg,S.L. (2015) Short read mapping: an algorithmic tour. In: *Proceedings of the IEEE*, pp, 1–23.

Cheng,H. *et al.* (2015) BitMapper: an efficient all-mapper based on bit-vector computing. *BMC Bioinformatics*, **16**, 192–207.

Consortium,G.P. (2012) An integrated map of genetic variation from 1,092 human genomes. *Nature*, **491**, 56–65.

David,M. *et al.* (2011) SHRiMP2: sensitive yet practical short read mapping. *Bioinformatics*, **27**, 1011–1012.

Hach,F. *et al.* (2010) mrsFAST: a cache-oblivious algorithm for short-read mapping. *Nat. Methods*, **7**, 576–577.

Hach,F. *et al.* (2014) mrsFAST-Ultra: a compact, SNP-aware mapper for high performance sequencing applications, *Nucleic Acids Res.*, gku370.

Hatem,A. *et al.* (2013) Benchmarking short sequence mapping tools. *BMC Bioinformatics*, **14**, 184.

Herbordt,M.C. *et al.* (2007) Achieving high performance with FPGA-based computing. *Computer*, **40**, 50.

Homer,N. *et al.* (2009) BFAST: an alignment tool for large scale genome resequencing. *PloS One*, **4**, e7767.

Houtgast,E.J. *et al.* (2015) An FPGA-Based Systolic Array to Accelerate the BWA-MEM Genomic Mapping Algorithm. In: *International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS)*. Samos Island, Greece.

Jacobsen,M. *et al.* (2015) RIFFA 2.1: a reusable integration framework for FPGA accelerators. *ACM Trans. Reconfigurable Technol. Syst.*, **8**, 1–23.

Langmead,B. and Salzberg,S.L. (2012) Fast gapped-read alignment with Bowtie 2. *Nat. Methods*, **9**, 357–359.

Langmead,B. *et al.* (2009) Ultrafast and memory-efficient alignment of short DNA sequences to the human genome. *Genome Biol.*, **10**, R25.

Levenshtein,V.I. (1966) Binary codes capable of correcting deletions, insertions, and reversals. *Soviet Phys. Doklady*, **10**, 707–710.

Li,H. (2013) Aligning sequence reads, clone sequences and assembly contigs with BWA-MEM, *arXiv preprint arXiv:1303.3997*.

Li,H. and Durbin,R. (2009) Fast and accurate short read alignment with Burrows–Wheeler transform. *Bioinformatics*, **25**, 1754–1760.

Li,M. *et al.* (2004) PatternHunter II: highly sensitive and fast homology search. *J. Bioinf. Comput. Biol.*, **2**, 417–439.

Lindner,M.S. *et al.* (2016) HiLive–real-time mapping of illumina reads while sequencing, *Bioinformatics*, btw659.

Liu,C.-M. *et al.* (2012) SOAP3: ultra-fast GPU-based parallel alignment tool for short reads. *Bioinformatics*, **28**, 878–879.

Luo,R. *et al.* (2013) SOAP3-dp: fast, accurate and sensitive GPU-based short read aligner. *PloS One*, **8**, e65632-e65632.

Marco-Sola,S. *et al.* (2012) The GEM mapper: fast, accurate and versatile alignment by filtration. *Nat. Methods*, **9**, 1185–1188.

Needleman,S.B. and Wunsch,C.D. (1970) A general method applicable to the search for similarities in the amino acid sequence of two proteins. *J. Mol. Biol.*, **48**, 443–453.

Olson,C.B. *et al.* (2012) Hardware acceleration of short read mapping. In: *Field-Programmable Custom Computing Machines (FCCM), 2012 IEEE 20th Annual International Symposium on IEEE*, pp. 161–168.

Rasmussen,K.R. *et al.* (2006) Efficient q-gram filters for finding all ε-matches over a given length. *J. Comput. Biol.*, **13**, 296–308.

Smith,T.F. and Waterman,M.S. (1981) Identification of common molecular subsequences. *J. Mol. Biol.*, **147**, 195–197.

Trimberger,S.M. (2015) Three ages of FPGAs: a retrospective on the first thirty years of FPGA technology. *Proc. IEEE*, **103**, 318–331.

Downloaded from https://academic.oup.com/bioinformatics/article/33/21/3355/3859176 by ETH Zurich user on 21 October 2021

Ukkonen,E. (1985) Algorithms for approximate string matching. *Inf. Control*, **64**, 100–118.

Ukkonen,E. (1992) Approximate string-matching with q-grams and maximal matches. *Theor. Comput. Sci.*, **92**, 191–211.

Waidyasooriya,H.M. *et al.* (2014) FPGA-Accelerator for DNA Sequence Alignment Based on an Efficient Data-Dependent Memory Access Scheme. In: *The international symposium on Highly-Efficient Accelerators and Reconfigurable Technologies (HEART2014)*. Sendai Miyagi, Japan, pp. 127–130.

Weese,D. *et al.* (2009) RazerS—fast read mapping with sensitivity control. *Genome Res.*, **19**, 1646–1654.

Weese,D. *et al.* (2012) RazerS 3: faster, fully sensitive read mapping. *Bioinformatics*, **28**, 2592–2599.

Xilinx. (2014) Virtex-7 XT VC709 Connectivity Kit, Getting Started Guide, UG966 (v3.0.1) June 30, 2014.

Xilinx. (2015) 7 Series FPGAs Overview, DS180 (v1.17) May 27, 2015.

Xin,H. *et al.* (2015) Shifted hamming distance: a fast and accurate SIMD-friendly filter to accelerate alignment verification in read mapping. *Bioinformatics*, **31**, 1553–1560.

Xin,H. *et al.* (2013) Accelerating read mapping with FastHASH. *BMC Genomics*, **14**, S13.

Downloaded from https://academic.oup.com/bioinformatics/article/33/21/3355/3859176 by ETH Zurich user on 21 October 2021