2018 IEEE International Symposium on High Performance Computer Architecture

Searching for Potential gRNA Off-Target Sites for CRISPR/Cas9 using Automata Processing across Different Platforms

Chunkun Bo, Vinh Dang, Elaheh Sadredini, Kevin Skadron
Department of Computer Science
University of Virginia
Charlottesville, USA
Email: chunkun, vinhdang, elaheh, skadron@virginia.edu

ABSTRACT

The CRISPR/Cas system is a bacteria immune system protecting cells from foreign genetic elements. One version that attracted special interest is CRISPR/Cas9, because it can be modified to edit genomes at targeted locations. However, the risk of binding and damaging off-target locations limits its power. Identifying all these potential off-target sites is thus important for users to effectively use the system to edit genomes. This process is computationally expensive, especially when one allows more differences in gRNA targeting sequences. In this paper, we propose using automata to search for off-target sites while allowing differences between the reference genome and gRNA targeting sequences.

We evaluate the automata-based approach on four different platforms, including conventional architectures such as the CPU and the GPU, and spatial architectures such as the FPGA and Micron's Automata Processor. We compare the proposed approach with two off-target search tools (CasOFFinder (GPU) and CasOT (CPU)), and achieve over 83× speedups on the FPGA compared with CasOFFinder and over 600× speedups compared with CasOT. More customized hardware such as the AP can provide additional speedups (1.5× for the kernel execution) compared with the FPGA. We also evaluate the automata-based solution using single-thread HyperScan (a high-performance automata processing library) on the CPU. HyperScan outperforms CasOT by over 29.7×. The automata-based approach on iNFAnt2 (a DFA/NFA engine on the GPU) does not consistently work better than CasOFFinder, and only show a slightly better speedup compared with single-thread HyperScan on the CPU (4.4× for the best case). These results show that the automata-based approach provides significant algorithmic benefits, and that accelerators such as the FPGA and the AP can provide substantial additional speedups. However, iNFAnt2 does not confer a clear advantage because the proposed method does not map well to the GPU architecture. Furthermore, we propose several methods to further improve the performance on spatial architectures, and some potential architectural modifications for future automata processing hardware.

1 Introduction

Clustered Regularly Interspaced Short Palindromic Repeats (CRISPR) exist in prokaryotic DNAs. The CRISPR/Cas system is an immune system which defends against foreign genetic elements [1]. CRISPR/Cas9 is one version of the system that attracts researchers' interest, because it can be modified to edit genomes [2]. One can deliver the Cas9 nuclease together with a guide RNA (gRNA) into a cell and edit the cell's genome at targeted locations defined by the gRNA. Genome editing using CRISPR/Cas9 has been a popular technique since it was first introduced. For example, researchers are trying to use cells edited by CRISPR/Cas9 to fight cancers [3]. The CRISPR/Cas9 system is also being used to cure other diseases with genetic causes, such as amnesia, muscular dystrophy, etc [4].

However, efficiently finding all correct locations to edit the genome, without modifying other locations, is still the bottleneck of using the CRISPR/Cas9 system, because the gRNA also binds to locations with slightly different DNA sequences [5]. This makes the process of finding all potential *off-target* sites (genome locations sufficiently similar to the gRNA targeting sequence) computationally expensive, especially when one allows more differences from the reference sequence. The time complexity of searching for exact matches is $O(n*l*L)$, where n is the number of gRNA targeting sequences to be searched, l is the length of the query sequence, and L is the length of the reference genome. The time complexity is even worse when differences are allowed. There are several local off-target search tools (running on local workstations), but the performance is not satisfying, even with the GPUs. It could take hours/days to search for potential off-target sites when only allowing a few mismatches [6]. The performance of web tools (providing web interfaces for users to provide gRNA tar-

2378-203X/18/$31.00 ©2018 IEEE
DOI 10.1109/HPCA.2018.00068

geting sequences and reference genomes) is even worse.

Existing CPU and GPU tools either are restricted to the number of mismatches or fail to return results due to the computational bottlenecks. As Moore's Law slows down, accelerators have attracted more interest. Spatial architectures such as the FPGA and Micron's Automata Processor (AP) provide better performance because they can process automata in massive parallelism by laying out a huge number of automata graphs directly in hardware. The FPGA can be flexibly configured by users to implement a specific algorithm (automata processing in this paper) with a large amount of logic blocks. REAPR [7] extended prior work on regular expressions to automata processing, allowing us to implement the automata-based method to search for gRNA off-target sites on the FPGA. The AP is an efficient architecture designed for parallel automata processing [8] [9], and has been used in many different domains such as association rule mining [10] [11], sequential pattern mining [12], tree mining [13], entity resolution [14], random forest [15], natural language processing [16], string kernel [17], pseudo-random number generator [**?**],etc. These applications require inexact matching against many patterns, which is similar to searching for potential gRNA off-targets.

In this paper, we propose an automata-based solution, to identify potential off-target sites by allowing any Hamming distances in a reference genome. We evaluate the proposed approach across four different platforms (CPU, GPU, FPGA and AP) and compare with two state-of-the-art solutions (CasOFFinder [18] and CasOT [19]). The proposed method leads to over 83× speedups on the FPGA compared with CasOFFinder (GPU) and additional speedups can be achieved by the AP. Furthermore, we evaluate the proposed automata-based method using HyperScan [20], a high-performance automata processing library for the CPU, as an alternative CPU implementation for the proposed approach. The results show that even with single-thread HyperScan, we still achieve promising results (over 29.7×) compared with CasOT (CPU). However, the newly proposed automata-based method with iNFAnt2 [21] on the GPU does not confer a clear advantage because it does not map well to the GPU architecture. These results show significant algorithmic benefits of the automata-based approach, and the potential speedups can be achieved by hardware acceleration for automata processing.

In summary, this paper makes the following contributions:

1. We propose an automata-based approach to search for potential gRNA off-target sites and implement the approach across four various platforms (CPU, GPU, FPGA, and AP).

2. We present several automata designs for searching for off-target sites with different requirements, such as different Hamming distances or mismatches in different regions of a potential sequence.

3. We evaluate the proposed automata-based method and compare against two state-of-the-art tools (CasOFFinder and CasOT). Promising speedups are achieved by the FPGA (over 83×) compared with CasOFFinder. Additional speedups could be achieved by more customized hardware such as Micron's AP.

4. We evaluate the automata-based method on the CPU using HyperScan and achieve over 29.7× speedups, showing the algorithmic benefit of the automata approach. However, the proposed method on the GPU does not confer a clear advantage, because it does not map well to the GPU architecture.

5. We discuss how to further improve the performance and support larger datasets on spatial architectures (the FPGA and the AP), and propose several potential architectural modifications for future automata processing hardware.

2 Related Work

Several methods have been proposed to search for potential gRNA off-target sites on local workstations. CasOFFinder is a fast off-target search tool [18]. It is written in OpenCL, making it portable across diverse platforms such as CPU and GPU. It supports an unlimited number of mismatches and different PAM sequences. However, it runs much slower as users allow more mismatches. CasOT is another popular potential off-target search tool [19]. It divides the targeting sequence into non-seed and seed regions, and allows different Hamming distances in each region. There are also no limits on how many mismatches are allowed in each region. Similarly, it runs slower as more mismatches are allowed. It could take more than a day to find a relatively large number of mismatches (e.g., larger than five). Some researchers use DNA alignment tools, such as PatMaN [22], Bowtie [23] or BWA-like algorithms [24], to search for potential off-target sites. However, these tools are not designed to solve this specific problem. They cannot recognize PAMs and the performance is not as good as CasOFFinder and CasOT. Furthermore, tools such as PatMaN and Bowtie have limitations on the number of allowed mismatches. Unlike these methods, the proposed method in this paper solves this search problem using automata processing, which allows any desired edit distance (we focus on Hamming distances in this paper) and can benefit from hardware architectures (FPGA and AP) to accelerate such computation.

Several automata processing methods on different platforms have been proposed. HyperScan is a high-performance automata matching library and uses hybrid automata techniques to match a large number of regular expressions simultaneously [20]. iNFAnt2, an optimized version of iNFAnt [25], is a prototype framework for NFA-based automata processing on NVDIA CUDA-enabled GPU cards [21]. iNFAnt2 can process very large and complex NFAs and DFAs with high throughput and low memory consumption [26] [27]. REAPR is a reconfigurable engine for automata processing on the FPGA. It provides a flexible framework which synthesizes RTL for applications involving automata processing with high throughput [7]. Two different implementations (logic-based and BRAM-based) are presented in

the REAPR paper, and we use the logic-based method to achieve better performance. Micron's AP is an efficient semiconductor architecture for parallel automata processing [8]. It uses a non-von-Neumann reconfigurable architecture, which directly implements NFA in hardware, to match complex regular expressions and other types of automata that cannot be conveniently expressed as regular expressions.

3 Automata Processing on Spatial Architectures

Automata Processing can be defined as a directed graph where the nodes store the states of the automata and the edges store the transition rules between states. To be specific, each node stores the symbols to be matched and reads the symbol from the input stream. All current active states compare with the next input symbol. When the node matches with the input symbol, it will activate all the nodes connected to it by the directed edges. Each automaton has at least one "start" state to initiate the processing and one or more "accept" states to report when a match is found in the input.

Spatial architectures can process certain computation using reconfigurable processing elements. For example, the FPGAs use logic gates and RAM blocks to implement a specific algorithm. Spatial architectures usually provide a large number of hardware resources. For example, the AP A480 board can store up to 1.5 million states. Therefore, the spatial architecture can process automata by placing the automata graphs directly in hardware instead of storing all states and transition rules in the memory as conventional architectures do. It reduces the cost of memory access on each input symbol to search for the next states to be activated. With massive hardware resources, the spatial architecture can process a large number of automata simultaneously.

4 CRISPR/Cas9 System

The CRISPR/Cas system was originally found in bacteria immune systems as a way to fight against foreign genetic elements [1]. We list the terminologies used in the paper in Table 1. When a foreign genetic element enters the cell, a Cas complex recognizes it and cleaves it into small fragments and adds a new spacer to the end of the CRISPR. The new array is then transcribed into an RNA, which will direct the Cas complex to the foreign DNA. The targeted DNA will then be destroyed or cleaved. This system was later applied in genome editing as shown in Figure 1 [2]. The gRNA contains the targeting sequence and works together with the Cas9 complex to bind to the reference genome. They bind to the specific location (complementary to the gRNA targeting sequence followed by a PAM) and can be further used to edit the target sequence. One major problem is that the gRNA and Cas9 do not bind only to the desired position, but also to other locations where some DNA characters are different from the sequence but close enough to bind, leading to the off-target effects [5]. In this paper, we aim to accelerate the process of identifying all these potential off-targets.

Term	Definition
CRISPR	Clustered Regularly Interspaced Short Palindromic Repeat. A region originally found in bacteria to defend foreign intrusions.
Cas	CRISPR Associated protein, the active enzyme in Type II CRISPR system.
gRNA	Guide RNA, works with Cas9 to target and bind to the specific location. It is also known as single guide RNA or sgRNA.
gRNA targeting sequence	The nucleotides before the PAM sequence, mostly 20 DNA characters.
Target sequence	The genome sequence the gRNA targets at, which can be incorporated into the gRNA and the PAM.
PAM	Protospacer Adjacent Motif, following the gRNA targeting sequence. Necessary for Cas9 to bind target sequence.
Off-target effects	Cas9 cleavages at unwanted locations because the gRNA binds to unintended locations with sequences similar to the gRNA.
Potential off-target sites	Locations in the reference genome that could be the off-target binding sites within a specified edit distance of the gRNA targeting sequence.
Reference genome	The genome the gRNA searches against.
Complementary reference genome	The reverse order of the reference genome, with complementary DNA characters.

Table 1: Terminology

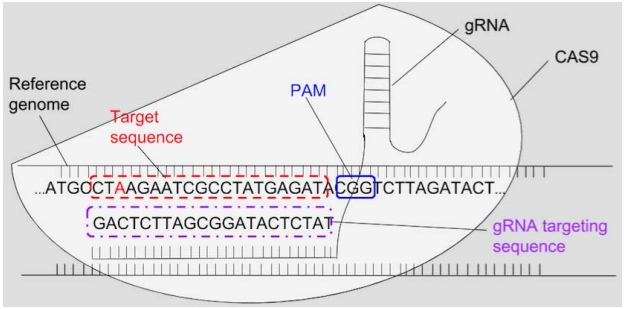

Figure 1: gRNA and CAS9 bind to the target sequence [28].

5 gRNA Off-target Sites Search using Automata Processing

In this section, we first present several general automata structures recognizing sequences with differences and

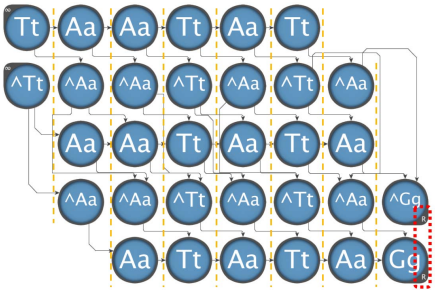

Figure 2: Hamming Distance Automaton.

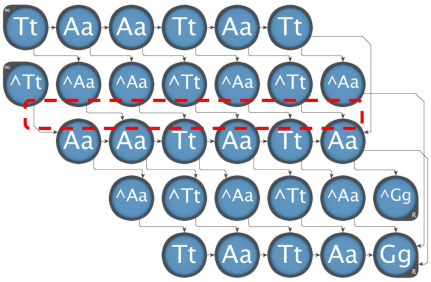

Figure 3: Hamming Distance Automaton with no consecutive mismatches.

use these structures to build the automata recognizing potential off-target sites. Later, we will show the general workflow of the automata-based method.

5.1 Hamming Distance Automaton

Figure 2 shows an automaton design recognizing the sequence "*TAATATAG*" when the Hamming distance is shorter than 3. There are five rows and eight columns in the example. The STEs in the ith column store the ith character in the sequence. The odd rows store the DNA characters one wants to match. Because there is no standard for whether uppercase or lowercase letters are used, this design stores both – both forms can be checked simultaneously, at no extra cost. The even rows match on any other symbol except for the desired character, thus capturing an increase in the Hamming distance. The STEs in odd rows connect to the next STE in the same row and the next STE in the next row. The STEs in even rows connect to the next STE in the next row and the next STE in the third row below them. This allows processing to proceed whether or not the current symbol matched. The structure can be extended to recognize sequences with more mismatches, but will consume more STEs. One needs $(2k+1)$ rows to recognize k mismatches. The STEs in the first column is configured as *all-input*, reading from any position in the input. As indicated in Figures 2, the STEs in the last column are configured as *reporting* and they will trigger a report when they are activated. This structure was also used to search for motifs [29]. However, the structure cannot be directly used to search for gRNA target off-target sites. It does not support specifying the regions of differences or identifying PAM sequences, but it can be used as a component to build a larger and more complex automaton to identify gRNA off-targets.

5.2 No Consecutive Mismatches Automaton

In Figure 3, we present another automata structure for Hamming distance, but it does not allow consecutive mismatches. In this example, we still allow two mismatches, but no two mismatches take place consecutively. This is particularly useful because researchers have discovered that if two mismatches take place consecutively, it is less likely the location containing the consecutive mismatches is a correct off-target site [30]. This design is similar to Figure 2, but as highlighted in the center of Figure 3, we do not connect the STE in the second row, where the mismatch takes place, to the STE in the fourth row. This assures that one mismatch can only be followed by a matching character. Two more STEs are eliminated at the bottom left corner.

The above examples show that the automata structures for recognizing Hamming Distances are straightforward. With small modifications, we are able to process complex queries. These are just two examples to illustrate how to build these structures. Users can modify these structures or propose a new structure to solve their specific problem. The differences are not limited to Hamming distance. For example, automaton for edit distance can be found in [31] and [32]. The rest of the section presents how we use these structures to build an automaton to identify gRNA off-target sites.

5.3 Mismatches in Whole Sequences

Depending on where mismatches take place, different automata designs can be used. For example, CasOFFinder allows the mismatch to take place in any position in the gRNA targeting sequence. Figure 4 represents the design for this requirement. The design is straightforward and we use the Hamming distance automata to match the gRNA targeting sequence with one mismatch allowed. We then connect the STEs in the last column in the gRNA targeting sequence to the start of the PAM sequence. No mismatches are allowed in the PAM sequence, so we just connect one STE to another. The STE storing the last character is the reporting STE. The STE storing "*" matches with any input symbol so we can match PAM sequences, such as "NGG" in streptococcus pyogenes, where "N" refers to any DNA character in {AaTtGgCc} in this context.

5.4 Mismatches in Different Regions

Some tools propose dividing the sequences into regions and allowing different Hamming distances in different regions. For example, CasOT divides the targeting sequence into a non-seed region (the first 8 characters) and a seed region (the next 12 characters) [6]. Users can specify different numbers of mismatches in these two regions. This may help to improve the quality of potential off-target sites, especially when scoring the potential off-targets. We show one example automaton for CasOT in Figure 5. This example allows 2 mismatches in the non-seed region and 1 mismatch in the seed re-

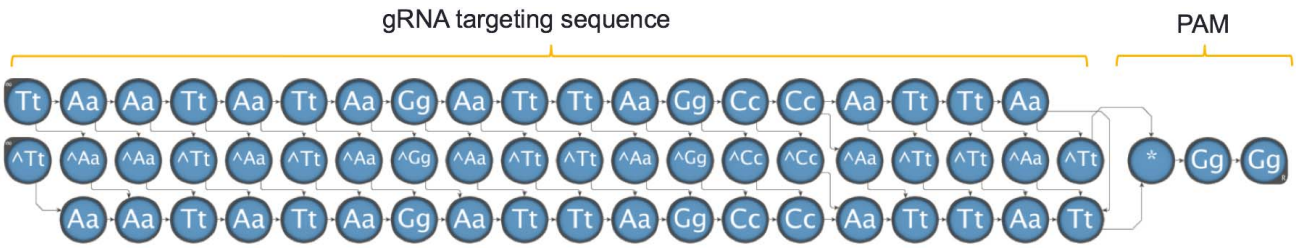

Figure 4: Allowing mismatches in any position.

gion. Both regions use the Hamming distance automata structure, but with different lengths and different distances. We connect the STEs in the last column in the non-seed region to the STEs in the first column in the seed region. We then connect these regions together. This is just one example of more complex queries. Users can divide the sequence into any number of regions, and apply different automata structures in each region.

5.5 Multiple PAMs

Users sometimes need to identify potential gRNA targeting sequences with different PAM sequences. This is straightforward using automata processing. We just connect the STEs storing the last DNA character in the gRNA targeting sequence, regardless of which automaton structure is used, to all the starting STEs in the PAM sequences (highlighted in Figure 6). For example, similar to what we described in Figure 4, we add two more PAM sequences ("NGCG" and "NNNNGATT") in Figure 6 and connect the STEs in the last column of the gRNA targeting sequence to the starting STEs in the new PAM sequences. The STEs storing the last DNA character in the PAM sequences are configured as reporting STEs. Because the reporting STEs have different IDs, we can differentiate which PAM sequence finds a match. For the current AP hardware, STEs can have up to 16 output connections, so we can recognize 16 different PAM sequences simultaneously. If one wants to recognize more than 16 PAM sequences, they need to duplicate the structure for the gRNA targeting sequence and connect to the remaining PAM sequences until all PAM sequences are processed.

5.6 Workflow

With the above automata, we can identify potential off-targets using automata processing. We will use the AP as an example to illustrate the workflow of the automata-based approach (Figure 7). The CPU first extracts the gRNA targeting sequences and PAMs from the database in a pre-processing step, because the original database may contain other information other than gRNA targeting sequences and PAMs. We then store these queries using the automata presented above on the AP board. The reference genome is streamed into the AP afterward and the AP compares the stored queries with the genome. If the AP finds a match, it reports. Based on the reporting STE ID and the offset of the reporting cycle, we can recognize which query finds a potential off-target, and at which position in the genome. Many existing tools support finding potential off-target sites for both strands of the reference genome, because of the double strand structure of DNA. If this is the case, we also need to stream the complementary reference genome to the AP. A reconfiguration phase is needed if the query number exceeds the capacity of the AP. The symbol replacement feature of the AP allows fast replacement of patterns, if the structure of the automata graphs remains unchanged. This feature makes the reconfiguration phase negligible (milliseconds), but one needs to stream in the reference genome again after the reconfiguration.

The workflow of REAPR on the FPGA is similar, but without the fast symbol replacement. Supporting REAPR with symbol replacement is left for future work. Users need to recompile for larger datasets, which may take a longer time to process large datasets. The workflow of iNFAnt2 is also similar, but users need to convert the input patterns to NFA/DFA files before streaming the reference genome to the GPU.

5.7 Fast Complementary Genome Processing

In Section 5.6, we discussed how to process the complementary reference genome and we used this method for evaluation in Section 6. An alternate method can be used to solve this by storing the complementary sequences "A"→"T", "T"→"A", "G"→"C", "C"→"G") of the gRNA targeting sequence in the automaton. This method works faster if one has a small set of gRNA targeting sequences and can store all the queries and complementary sequences on one FPGA board or one AP board, because we do not need to stream the complementary reference genome to the hardware. The example in Figure 8 recognizes the complementary sequence "TCTAGAGC-TCTAGAGCAGTA-NGG") in Figure 5. This structure connects the complementary PAM sequence to the complementary seed region and then the complementary non-seed region. The first STE in the complementary PAM sequence is the new starting STE and the last two STEs in the complementary non-seed region are the reporting STEs. All the STEs store the complementary DNA character in the original sequence in reverse order, so the new sequence to be recognized is "CCN-TACTGCTCTAGA-GCTCTAGA".

All these automata structures described in the above sections can be used by HyperScan on the CPU, REAPR on the FPGA, iNFAnt2 on the GPU, and Micron's AP.

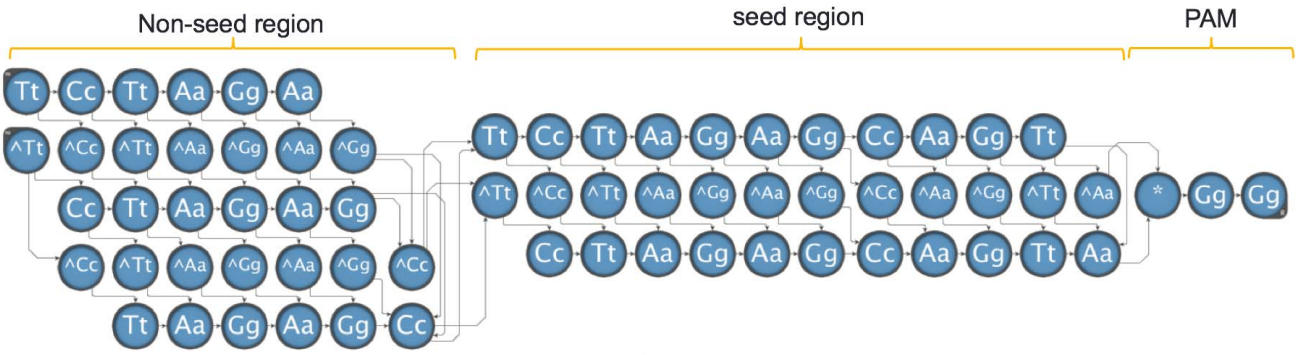

Figure 5: Allowing mismatches in different regions.

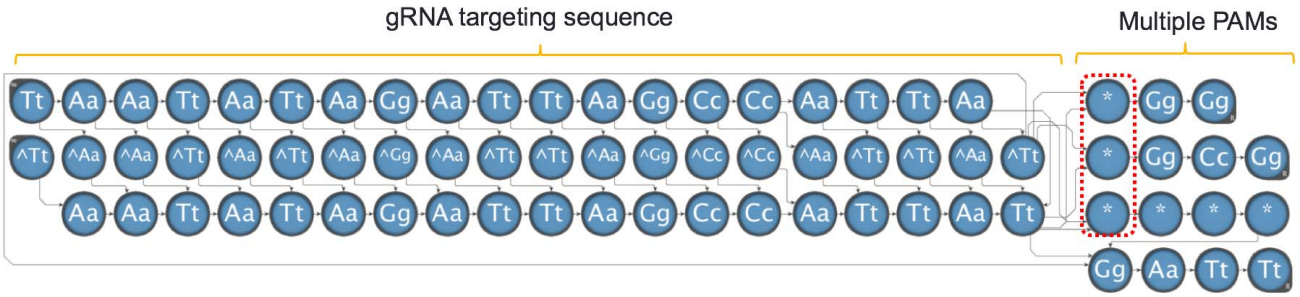

Figure 6: Allowing mismatches in any position with multiple PAM sequences.

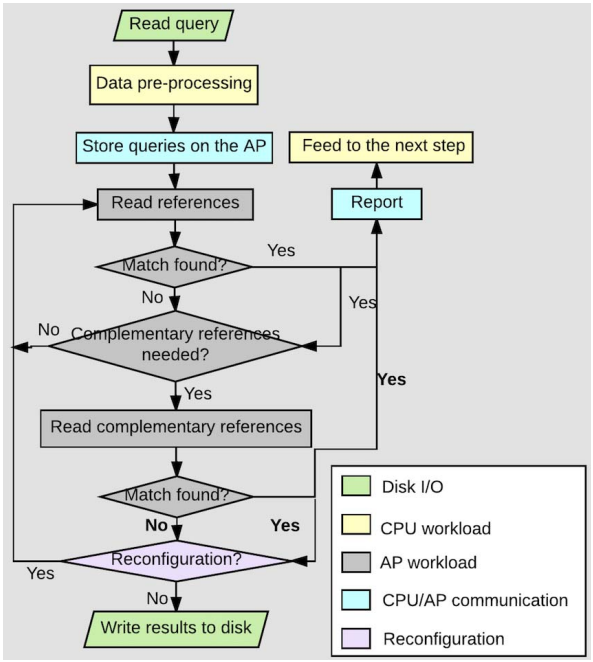

Figure 7: Workflow on Micron's AP.

6 Performance Evaluation

To evaluate the performance of the proposed approach, we compare with CasOFFinder (GPU) and CasOT (CPU). We implement the automata approach to match the tasks that CasOT and CasOFFinder solve, so they provide the same accuracy. Four different platforms are evaluated: HyperScan on the CPU, iNFAnt2 on the GPU, REAPR on the FPGA, and Micron's AP. The experiments are executed on a server with an Intel Core i7-5820K CPU (3.3GHZ) with 32GB RAM and an NVIDIA Tesla K40c GPU with 12GB memory. For FPGA results, we use an Alpha Data ADM-PCIE-KU3 board equipped with a Xilinx Kintex UltraScale XCKU060 FPGA and two 8GB DDR3 memory banks, hosted in a system with an Intel Core i7-4820K CPU (3.7GHz) with 32GB RAM. All results are actual runtimes except for the AP, because the AP hardware is not yet available. But it is simple to estimate the kernel execution time on the AP, because the input processing rate is fixed at 133MB/s (one character per cycle). The number of queries that can be placed on the board, assuming a 32-chip Micron's D480 AP board, determines how many passes through the input are required.

The human genome is used as the reference genome (3.2 billion base pairs). We develop two different generators to produce gRNA targeting sequences plus PAM sequences, because CasOT and CasOFFinder have different input formats and have various specifications of where to allow mismatches. We use these generators to study how numbers of queries and mismatches affect the performance. These synthesized datasets are only used to study the performance projections. One can use real datasets to study the biological relationship between the reference genome and the gRNA targeting sequence using the proposed method. More details of the generators will be presented in the following sections.

In this section, we use the total runtime and the ker-

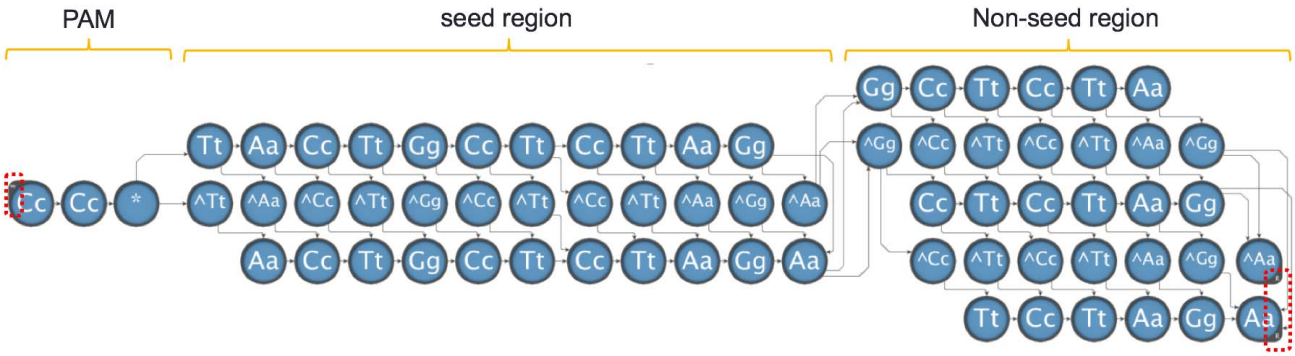

Figure 8: Process complementary sequence.

nel execution time (the time used for recognizing off-target sites) as criteria. We compare the performance with CasOFFinder and CasOT separately because they use different input formats and allow mismatches in different regions in gRNA targeting sequences.

6.1 CasOFFinder

CasOFFinder is a fast off-target search tool written in OpenCL, portable across the CPU and the GPU. We compare with CasOFFinder on the GPU to achieve its best performance. We use the design in Figure 4, because CasOFFinder does not isolate the position in gRNA targeting sequences. The generator for CasOFFinder generates the input sequences based on the format specifications. The first line stores the location of the reference genome. The second line is the desired gRNA targeting sequences along with PAM sequences. The sequences are followed by the Hamming distance. The first 20 characters of each query are the gRNA targeting sequence and are selected randomly from DNA character set {AaTtGgCc}, because no real large datasets are available. Without loss of generality, the synthesized datasets help us to study the performance of different approaches on various platforms. The PAM sequence and the number of mismatches are specified by users. We use "NGG" as an example of the PAM sequence to illustrate the suitability of the proposed automata-based method. We presented how to support multiple sequences in Section 5.5.

6.1.1 Spatial Architectures

Figure 9 and Figure 10 show that spatial architectures such as the FPGA, consistently outperform CasOFFinder by at least 14× for both large number of queries and long Hamming distances.

In Figure 9, we present the total runtime and the kernel execution time for REAPR. REAPR works much faster than CasOFFinder for all datasets. The runtime of REAPR depends heavily on the number of queries. Many reports could be generated for each symbol, and these reports need to be preserved for post-processing on the CPU. This massive data transfer, especially from the reports, has a significant impact on the overall performance on the FPGA. REAPR shows better kernel execution performance than the AP for query numbers (e.g. 30 seconds for 100 queries and 40 seconds for 500 queries). However, for large query numbers (more than 1,000 queries), the AP starts to work faster. We only show the results of queries fewer than 1,000 on the FPGA because of the limitation of the maximum supported memory bitwidth (i.e. 512 bits) of the FPGA system we use. We will discuss how to process large datasets on the FPGA in Section 7.1.1. For the AP, as long as we can store all the queries on one AP board, the kernel execution time stays constant (48 seconds). Processing the reference genome and the complementary reference genome each takes 24 seconds. Since the AP connects to the host CPU by PCIe interface as the FPGA board does, if we assume the AP uses the same report architecture as the FPGA does, we could achieve another 1.9× speedup for the total runtime compared with REAPR. The runtimes of CasOFFinder and single-thread HyperScan increase linearly as query numbers increase. CasOFFinder is only 2.1× faster than single-thread HyperScan for the best case.

In Figure 10, we show the runtimes for different Hamming distances. We choose two examples (n = 500 and 1,000); similar results are found for other query numbers. For REAPR, both the kernel execution time and the total runtime stays constant as we allow more mismatches for a specific number of queries since it only depends on the number of queries. However, the runtimes of CasOFFinder and HyperScan increase as more mismatches are allowed. For a smaller number of mismatches (m = 1 and m = 2), HyperScan is almost as fast as CasOFFinder. CasOFFinder starts to run faster as more mismatches are allowed, but only slightly better. As for the AP, no matter how many mismatches we allow, as long as we can store the queries on one AP board, the kernel execution time stays constant. In our experiments, we allow 5 mismatches at most, because when allowing more than 5 mismatches, CasOFFinder works too slowly for us to obtain runtimes. However, the gRNA also binds to locations with more than 5 mismatches, in which situation CasOFFinder cannot serve its intended purpose. Table 2 shows the maximum sequence numbers we can store on one AP board when allowing different mismatches using Micron's AP compiler. The possible speedups are also shown presented assuming the AP use the same reporting architecture

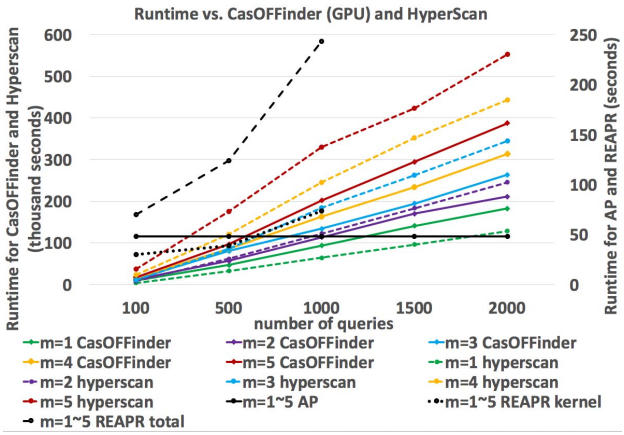

Figure 9: Runtimes vs. CasOFFinder for different numbers of queries. m is the number of mismatches. Dotted lines are runtimes of HyperScan and solid lines are runtimes of CasOFFinder. The lines with the same color refer to the same number of mismatches. Black lines represent the results of the AP and REAPR.

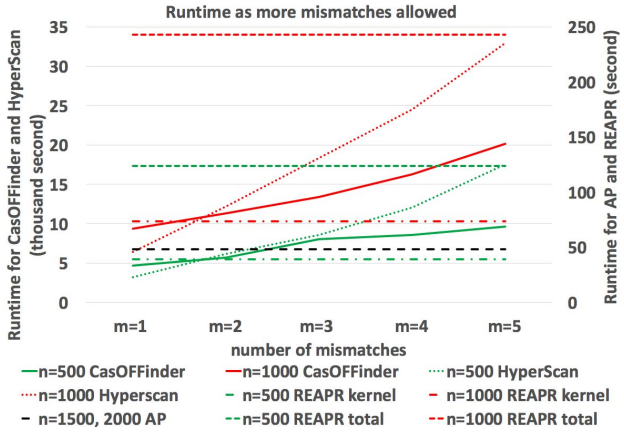

Figure 10: Runtimes vs. CasOFFinder for different mismatches (m). Lines with the same color refer to the same query number. The black line represents the runtimes for the AP.

as the FPGA. Compared to the total runtime of CasOFFinder, the AP could run over 180× faster. "NA" in the table refers to the cases when CasOFFinder works too slowly and does not return results in ten hours, implying the speedup is even larger. The AP works especially well when comparing a large number of queries with many mismatches allowed.

6.1.2 *iNFAnt2 on GPU*

As Figure 11 shows, the automata-based method on the GPU with iNFAnt2 (both the NFA engine and the DFA engine) does not confer a clear advantage compared with CasOFFinder. While the NFA engine does achieve a speedup of 2.4× where n = 500 and m = 1, it is slower by 1.7× where n = 2,000 and m = 5. For smaller query numbers or shorter Hamming distances, the DFA engine

Mismatches	1	2	3	4	5	6
Max number (K)	22.0	12.6	7.2	5.0	3.4	2.0
Speedup nq=500	53	65	91	98	110	NA
Speedup nq=1,000	73	89	105	127	157	NA
Speedup nq=2,000	88	102	127	150	186	NA

Table 2: Max queries stored on one AP board and possible speedups against CasOFFinder for different numbers of mismatches. nq is the number of queries."NA" refers to the cases when CasOFFinder does not finish within 10 hours.

works faster than the NFA engine. However, because of the state explosion of the DFA for large automata, the DFA engine does not work for larger pattern numbers or longer Hamming distances. For example, the DFA engine stops working when n = 100, m > 4, because the DFA table size is too large and exceeds the capacity of the GPU memory (12GB).

iNFAnt2 (DFA and NFA) only works faster than single-thread HyperScan on the CPU for some cases. For the best case, iNFAnt2 (DFA) is 4.4× faster (n = 2,000, m = 1), and iNFAnt2 (NFA) is 1.7× faster (n = 500, m = 1). This is because both NFA and DFA engines in iNFAnt2 suffer from the divergence problem. For NFA engine, for each input symbol, multiple CUDA threads within a thread block are launched to examine all possible transitions corresponding to that symbol across every state in the automata. For the automata used to search for gRNA off-target sites, we only use four DNA characters (both upper and lower cases) plus the "*" character, but the number of transitions on each input character is large, which requires a large number of sequential thread-block iterations. Despite the large number of transitions for each symbol, the number of active states is relatively small (approximate 20-30× smaller than the number of the examined transitions), which potentially causes degraded performance due to thread divergence. For DFA engine, we assign each independent automaton (pattern) to its own CUDA thread. Even if there are enough automata to fill the threads, their memory access for transition lookup and matching behavior are divergent. Therefore, the automata-based approach does not map well to the GPU architecture.

The above results clearly show the suitability and advantage of the proposed automata-based method using spatial architectures, such as the FPGA and the AP. Furthermore, compared with CasOFFinder (GPU), the method using automata processing with single-thread HyperScan on the CPU is only slightly slower, implying the automata-based method provides substantial algorithmic benefit. iNFAnt2 only provides a limited advantage over CasOFFinder, because the automata-based approach is not well mapped to the GPU architecture.

6.2 CasOT

CasOT is an off-target search tool on the CPU. It divides the targeting sequence into non-seed and seed regions and allows different Hamming distances in each region. Therefore, we use the design in Figure 5 to com-

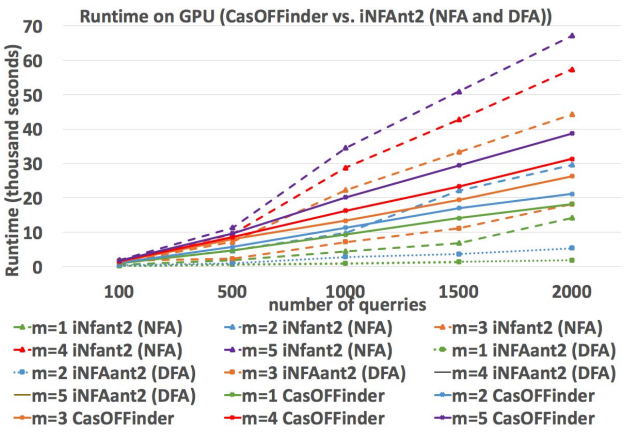

Figure 11: Runtimes of iNFAnt2 (NFA & DFA) for different query numbers. m is the number of mismatches. Dotted lines are runtimes of the NFA engine and solid lines are runtimes of the DFA engine. The lines with the same color refer to the same number of mismatches.

pare with CasOT. The generator for CasOT produces two lines for each query. The first line is the name of the sequence and the second line is the gRNA targeting sequence followed by the PAM sequence. We generate the first 20 characters randomly from DNA character set {AaTtGgCc} of each query and use "NGG" as an example for the PAM sequence.

6.2.1 *Spatial Architectures*

Figure 12 and Figure 13 show that the automata-based method on the FPGA always works faster than CasOT for different query numbers and Hamming distances (over 29.7× speedups).

In Figure 12, the total runtimes of REAPR increase as the query number increases, and REAPR is at least 7.5× faster than CasOT. Higher speedups are achieved for larger datasets. Since the bitwidths of the memory interface currently supported on the FPGA are 32, 64, 128, 256, and 512 bits, the runtime for one particular number of queries will be governed by the I/O bitwidth closest to it (e.g. 500 queries to 512 bits). REAPR shows better kernel execution performance than the AP for small numbers of queries ($\leq$ 512 queries), while the AP outperforms the REAPR for larger numbers of queries. For the AP, since we can store all the queries on one AP board, the kernel execution time stays constant (48 seconds). The total runtimes of CasOT and HyperScan increase linearly as the query number increases. For a small number of queries, CasOT runs faster than HyperScan. However, as the query number increases, the runtime of CasOT increases significantly. Many data points are missing in this figure for CasOT, because it runs too slowly and cannot return results in 24 hours. HyperScan beats CasOT for most of the cases and over 29.7× speedup is achieved based on the results we can get from CasOT. Figure 12 seems to suggest that HyperScan has the highest runtimes, but this is because CasOT cannot even run for larger query numbers. Therefore, speedups of HyperScan are even greater for such cases. We also evaluate the automata-based method on iNFAnt2, but only get 1.88× speedup for the best case compared with HyperScan (n = 700 and m = 1), similar to the results in Section 6.1.

Figure 13 show the performance for different Hamming distances. We only compare small query numbers (50 and 100), because CasOT runs extremely slowly for larger datasets. The total runtime and the kernel execution time of the FPGA stay constant; while the runtime of CasOT increases almost exponentially as more mismatches are allowed. Compared to what we can get from CasOT, we achieve over 600× speedups on the FPGA. The kernel execution time of the AP also stays constant (48 seconds). Table 3 shows the maximum query numbers we can store on one AP board for different numbers of mismatches. We also present the possible speedups assuming the AP shares the same reporting architecture as the FPGA does.

The results prove the great advantage brought by the automata-based method on the FPGA, and potential speedups can be achieved by the AP. If a hardware accelerator is not available, HyperScan on the CPU also provides substantial speedups compared with the existing methods on the CPU. However, iNFAnt2 on the GPU only provides minimal speedups compared with the single-thread HyperScan on the CPU.

In summary, the automata-based method running on spatial architectures, such as the FPGA and the AP, work consistently faster than the state-of-the-art tools, especially when comparing larger numbers of queries or allowing larger numbers of mismatches. Promising speedups are achieved compared with CasOFFinder on the GPU (over 83× on the FPGA) and even better speedups are achieved compared with CasOT on the CPU. The more customized hardware such as the AP could provide additional speedups compared with the FPGA. High speedups are achieved by the spatial architectures because they can implement a large number of automata storing the queries directly in hardware and process these queries simultaneously. Furthermore, the method using single-thread HyperScan also achieves promising results (over 29.7× speedups compared to CasOT) and only works slightly slower than CasOFFinder. This shows that the automata approach confers a significant algorithmic advantage, especially for large numbers of queries and mismatches. The differences in speedups between HyperScan and the FPGA/AP show the benefits of hardware acceleration for automata processing, e.g., 88× faster when searching 200 queries with 6 mismatches allowed in the seed region on the FPGA. However, the current automata-based method on the GPU only provides a minimal advantage over the CPU, because the automata processing approach does not map well to the GPU architecture.

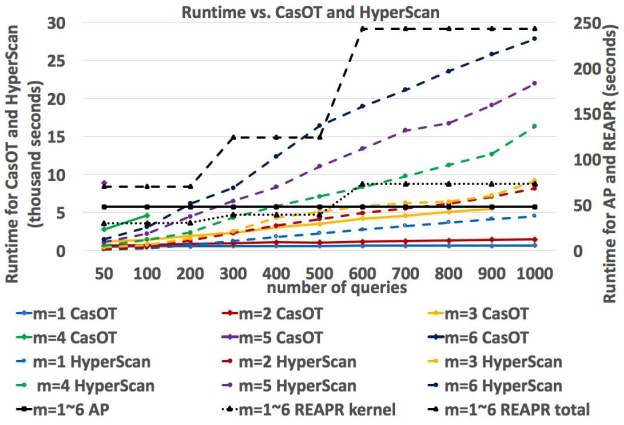

Figure 12: Runtimes vs. CasOT for different numbers of queries, where m is the number of mismatches. Dotted lines are runtimes of HyperScan and solid lines are runtimes of CasOT. The lines with the same color refer to the same number of mismatches. Black lines represent the results of the AP and REAPR. Many data points of CasOT are missing because it cannot return the results in 24 hours.

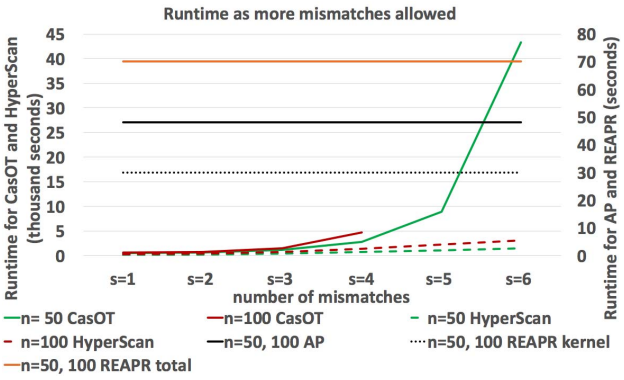

Figure 13: Runtimes vs. CasOT for different mismatches(m).

Mismatches	1	2	3	4	5	6
Max number (K)	26.2	16.5	11.8	8.3	5.8	3.2
Speedup nq=50	6	7	13	31	101	492
Speedup nq=500	7	12	40	NA	NA	NA
Speedup nq=1,000	5	11	NA	NA	NA	NA

Table 3: Max number of queries stored on one AP board and possible speedups for different numbers of mismatches. nq is the number of queries."NA" refers to the cases when CasOT cannot finish within 24 hours.

7 Further Improvement on Spatial Architectures

7.1 FPGA

7.1.1 Support Large Datasets

The computation core of REAPR is the automata processing module where state-transition elements are mapped to registers and lookup tables on the FPGA [7]. REAPR takes advantage of Xilinx SDAccel [33] to generate PCIe and AXI circuitry for the I/O interface to transfer data to and from the automata processing kernel. The entire design, including the automata processing module and the I/O circuitry, is laid out on the reconfigurable fabrics. Hence, the main limitations of REAPR are the FPGA capacity and the memory interface. Because of the maximum memory bitwidth (512 bits) currently supported by SDAccel, we implement the 1024-bit report architecture using two memory ports, each of which is attached to a DDR bank on the FPGA KU3 board with two DDR banks. Furthermore, when we search for 1,000 queries and allow 5 mismatches, the design already utilizes around 90% of the hardware resource. Due to these two reasons, the current REAPR implementation in this paper can support up to 1024 queries (i.e. 1024 reporting states in the NFA).

Larger FPGA boards provide more hardware resources, and thus relieve the bottleneck. There are also several other possible solutions. One is using a compressed report architecture to reduce the reports size so that they fit into maximum bitwidth. We profiled the results and found there are only a few reports in the same cycle, making the report vector fairly sparse and could be compressed by a lot. Assuming we compress the report to a quarter of its original size, we can process four times more queries. Another solution is using an on-chip buffer to temporarily store the reports. Instead of transferring the reports back to the CPU immediately, it will not report until the buffer is full. A double-buffering architecture would be useful to improve the throughput for this solution.

7.1.2 Higher Parallelism

Approaches can be applied to further improve the parallelization. The current automata processing kernel on REAPR is synthesized on a single compute unit (CU, the element in the FPGA device on which the kernel is executed). It would be helpful to exploit the parallelism among the independent automata and among the automata input streams on multiple CUs to increase throughput. Specifically, distinct automata can be divided into a number of sub-automata, and the input stream can be divided into chunks properly. We then synthesize multiple CUs on the FPGA device to process these sub-automata and chunks simultaneously. This helps to further reduce the total runtime, as long as there are enough resources on the FPGA device. For the multi-CU implementation, each CU should be assigned to a separate set of memory banks for optimum memory access. Multiple CUs sharing the same memory bank may lead to performance degradation because of memory access contention. This model can be implemented on a multi-CU, multi-FPGA architecture on a local workstation or even in cloud-based FPGA platforms such as the Amazon Web Services F1 [34] or Nimbix Cloud [35]. For instance, we implemented the multi-CU design on a larger board (Xilinx XILACCEL-RD-KU115 board equipped with a Kintex UltraScale

Table 4: Runtimes for large datasets when 3 mismatches are allowed.

Query Number(k)	≤7.2	7.2-14.4	14.4-21.6	21.6-28.8	28.8-26.0	26.0-43.2	43.2-50.4
Runtime (seconds)	48	96.045	144.09	192.135	240.18	144.355	168.27

XCKU115 FPGA and four DDR4 banks). We access the KU115 board through Nimbix Cloud. The XCKU115 FPGA board is 2× larger than the XCKU60 FPGA board used in our experiments and enables synthesizing kernels work at 300MHz (250MHz for the KU3 board). Taking all these factors into account, we can achieve another 2.4× speedup compared to the results in the previous section.

7.2 AP

7.2.1 Support Large Datasets

For larger datasets, the number of STEs is the bottleneck of current AP board. The number of queries that can be processed increases linearly as the number of STEs increases. In this section, we discuss how to process large datasets on the existing AP board, and use the design in Section 5.3 as an example. Table 4 presents the kernel execution time when the Hamming distance is 3 and the query number is larger than $12,000$ (maximum queries on one AP board). The symbols stored in the STE can be replaced quickly if the automata graphs stay unchanged, and it takes 45 milliseconds to reconfigure the whole AP board. As shown in the workflow (Figure 7), we need to stream in the reference/complementary genome after the symbol replacement. The kernel execution time in the table includes the time of streaming the reference/complementary genome and the reconfiguration time. However, compared with the streaming time, the reconfiguration time is almost negligible. The kernel execution time of the AP increases almost linearly as the query number increases, in steps corresponding to the number of symbol replacement passes. The time will not increase until another reconfiguration and another pass of the input are needed (when the query number is larger than $24,000$). The clock speed, and thus the rate at which the reference genome, is processed is the largest factor in AP performance. Additional performance improvements can be achieved with hardware supporting higher clock speed.

7.2.2 Potential Architectural Modifications

Boards with Fewer Chips: The AP D480 board has 32 chips on one board. The gRNA targeting sequence is usually short (20-30). Therefore, the automata designs proposed to recognize these sequences do not consume many STEs. As a result, the AP can process a huge number of queries simultaneously. However, when users only want to search for a few queries, the AP board is underutilized. It might be more cost-effective for some markets to have smaller boards with fewer chips for small problem sizes.

Multiple Streams to Different Chips: Instead of building smaller boards, allowing different input streams to different chips also helps to better utilize the existing board. For example, if we store different queries on different chips and stream different reference genomes to different chips, we can get the results for different genomes simultaneously. We can also apply a similar method as described in Section 7.2.1: store the same set of queries on different chips, divide the reference genome properly, and stream the divided genome chunks to different chips. This helps to further reduce the runtime and better utilize the hardware.

Flexible Symbol Set Size: The current AP stores 256 different symbols in one STE. However, when searching for gRNA off-target sites, we only use 9 of them (four DNA characters including lowercase and uppercase and '*'). This is a common case in many bioinformatics applications. It is a waste of area and power to operate an STE storage array supporting a large symbol set without actually using it. Therefore, if one can reduce the symbol set size and use the saved area for more STEs, the hardware can store more patterns, thus leading to higher parallelism. This is beneficial for problems such as Next Generation Sequencing.

8 Conclusions and Future Work

In this paper, we proposed an automata-based approach for identifying potential gRNA off-targets, and presented several automata designs to solve this problem. To evaluate the suitability of the proposed method, we compared the automata-based approach on different platforms (CPU, GPU, FPGA and AP) with the state-of-the-art solutions (CasOT and CasOFFinder). Promising speedups are achieved (over 600× on the FPGA) and additional speedups could be achieved by the more customized hardware (AP). Based on the results of single-thread HyperScan (CPU) compared with CasOFFinder and CasOT, we conclude the automata-based method provides significant algorithmic benefits. However, the automata-based approach on the GPU provides a minimal advantage over CasOFFinder because it is not well mapped to the GPU architecture. Finally, we discussed how to further improve the performance (support larger datasets or high parallelism) on spatial architectures and proposed several potential architectural modifications for the future automata processing hardware.

Future work includes improving the proposed approach on the FPGA, extending to other applications, implementing proposed architectural modifications and enhancements to automata processing architectures.

9 Acknowledgements

This work was supported in part by grants from the NSF (CCF-1629450), and support from C-FAR, one of six centers of STARnet, a Semiconductor Research Corporation program sponsored by MARCO and DARPA.

10 References

[1] Horvath, P. et al. CRISPR/Cas, the immune system of bacteria and archaea. *Science*, 327(5962):167–170, 2010.

[2] Zhang, F. et al. CRISPR/Cas9 for genome editing: progress, implications and challenges. *Human Molecular Genetics*, 2014.

[3] Platt, R. J. et al. CRISPR-Cas9 knockin mice for genome editing and cancer modeling. *Cell*, 159(2):440–455, 2014.

[4] *11 Crazy Gene-Hacking Things We Can Do with CRISPR*. `http://www.popularmechanics.com/science/a19067/11-crazy-things-we-can-do-with-crispr-cas9/`.

[5] Fu, Y. et al. High-frequency off-target mutagenesis induced by CRISPR-Cas nucleases in human cells. *Nature Biotechnology*, 31(9):822–826, 2013.

[6] *CasOT - CRISPR/Cas system (Cas9/gRNA) Off-Targeter*. `http://casot.cbi.pku.edu.cn`.

[7] Xie, T. et al. Reapr: Reconfigurable engine for automata processing. In *The International Conference on Field-Programmable Logic and Applications (FPL)*, 2017.

[8] Dlugosch, P. et al. An efficient and scalable semiconductor architecture for parallel automata processing. *IEEE Transactions on Parallel and Distributed Systems*, 2014.

[9] Wang, K. et al. An overview of micron's automata processor. In *Proceedings of the IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis (CODES)*, 2016.

[10] Wang, K. et al. Association rule mining with the Micron Automata Processor. In *Proceedings of the IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, 2015.

[11] Wang, K. et al. Hierarchical pattern mining with the Automata Processor. *International Journal of Parallel Programming*, pages 1–36, 2017.

[12] Wang, K. et al. Sequential pattern mining with the Micron Automata Processor. In *Proceedings of the ACM International Conference on Computing Frontiers*, pages 135–144. ACM, 2016.

[13] Sadredini, E. et al. Frequent subtree mining on the automata processor: challenges and opportunities. In *Proceedings of the International Conference on Supercomputing (ICS)*. ACM, 2017.

[14] Bo, C. et al. Entity resolution acceleration using the Automata Processor. In *Proceedings of the IEEE International Conference on Big Data*. IEEE, 2016.

[15] Tracy II, T. et al. Towards machine learning on the automata processor. In *International Conference on High Performance Computing*. Springer, 2016.

[16] Zhou, K. et al. Brill tagging on the Micron Automata Processor. In *Procedings of the IEEE International Conference on Semantic Computing (ICSC)*, 2015.

[17] Bo, C. et al. String kernel testing acceleration using the Micron Automata Processor. In *Proceedings of the Workshop on Computer Architecture for Machine Learning (CAMEL)*, 2015.

[18] Bae, S. et al. Cas-OFFinder: a fast and versatile algorithm that searches for potential off-target sites of Cas9 RNA-guided endonucleases. *Bioinformatics*, page btu048, 2014.

[19] Xiao, A. et al. CasOT: a genome-wide Cas9/gRNA off-target searching tool. *Bioinformatics*, page btt764, 2014.

[20] *Hyperscan*. `https://01.org/hyperscan`.

[21] Wadden, J. et al. Anmlzoo: a benchmark suite for exploring bottlenecks in automata processing engines and architectures. In *IEEE International Symposium on Workload Characterization (IISWC)*. IEEE, 2016.

[22] Prüfer, K. et al. PatMaN: rapid alignment of short sequences to large databases. *Bioinformatics*, 24(13):1530–1531, 2008.

[23] Langmead, B. et al. Bowtie: an ultrafast memory-efficient short read aligner. *Genome Biol*, 10(3):R25, 2009.

[24] Li, H. et al. Fast and accurate short read alignment with burrows–wheeler transform. *Bioinformatics*, 25(14):1754–1760, 2009.

[25] Cascarano, Niccolo. et al. infant: Nfa pattern matching on gpgpu devices. *ACM SIGCOMM Computer Communication Review*, 2010.

[26] *iNFAnt2*. `https://github.com/vqd8a/iNFAnt2`.

[27] *DFAGE*. `https://github.com/vqd8a/DFAGE`.

[28] Zhang, X. et al. Off-target effects in CRISPR/Cas9-mediated genome engineering. *Molecular Therapy-Nucleic Acids*, 4:e264, 2015.

[29] Roy, I. et al. Finding motifs in biological sequences using the Micron Automata Processor. In *Proceedings of the IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, 2014.

[30] Singh, R. et al. Cas9-chromatin binding information enables more accurate CRISPR off-target prediction. *Nucleic Acids Research*, 43(18):e118–e118, 2015.

[31] *Levenshtein in ANMLZoo*. `https://github.com/jackwadden/ANMLZoo/tree/master/Levenshtein`.

[32] Yu, X. et al. Robotomata: A framework for approximate pattern matching of big data on an automata processor. In *Proceedings of the IEEE International Conference on Big Data (BigData)*. IEEE, 2017.

[33] *SDAccel*. `https://www.xilinx.com/products/design-tools/software-zone/sdaccel.html`.

[34] *Amazon EC2 F1 Instances*. `https://aws.amazon.com/ec2/instance-types/f1/`.

[35] *SDAccel on the Nimbix Cloud*. `https://www.nimbix.net/xilinx/`.