Exploration of Short Reads Genome Mapping in Hardware

Edward Fernandez, Walid Najjar
Department of Computer Science and Engineering
University of California, Riverside
Riverside, California, USA

Elena Harris, Stefano Lonardi
Department of Computer Science and Engineering
University of California, Riverside
Riverside, California, USA

Abstract— **The newest generation of sequencing instruments, such as Illumina/Solexa Genome Analyzer and ABI SOLiD, can generate hundreds of millions of short DNA "reads" from a single run. These reads must be matched against a reference genome to identify their original location. Due to sequencing errors or variations in the sequenced genome, the matching procedure must allow a variable but limited number of mismatches. This problem is a version of the classic approximate string matching where a long text is searched for the occurrence of a set of short patterns. Typical strategies to speed up the matching involve elaborate hashing schemes that exploit the inherent repetitions of the data. However, such large data structures are not well suited for FPGA implementations. In this paper we evaluate an FPGA implementation that uses a "naive" approach which checks every possible read-genome alignment. We compare the performance of the naive approach to popular software tools currently used to map short reads to a reference genome showing a speedup of up to 4X over the fastest software tool.**

Keywords-component: Reconfigurable computing, bioinformatics, string-matching,

I. INTRODUCTION

The newest generation of sequencing instruments, such as Illumina/Solexa Genome Analyzer [1] and ABI SOLiD [8], produce hundreds of millions of short DNA reads from a single three to four days run. The length of the reads currently ranges between 16 and 40 characters in length, but is progressively increasing with improvements in sequencing technology. The reads must be mapped to a reference genome allowing a limited but variable number of mismatches due to sequencing errors and variations in the sequenced genome. The size of the reference genomes ranges from 10^6 to 10^9 characters. The mapping problem is essentially a version of the classic approximate string-matching problem. It can become computationally challenging due to the size of the genome and the large number of reads generated by the sequencing instruments. It can be, however, parallelized and the basic problem of string matching has been shown to be well suited for FPGA implementations. However, proposed string matching approaches, outside of bioinformatics, do not consider approximate matches. The existing software tools rely on sophisticated hashing schemes that exploit the inherent repetitions in the genome data. Such large data structures, however, are not very well suited for FPGA implementations.

In this paper we develop and evaluate a "naive" approach using FPGAs that checks all possible alignments between the reads and the reference genome. The genome is streamed one character at a time per stream through the FPGA and matched against a subset of the reads. Since this algorithmic approach is highly parallelizable, we evaluate variants where the genome is split into multiple parallel streams. Experimental results show that two to four streams maximize the throughput of the FPGA.

In this approach, the hardware architecture is independent of the input reads and the target genome. The read data is loaded into the FPGA structure before initiating the streaming of the data. No FPGA reconfiguration is required among different sets of inputs.

We compared the performance of the naïve method to that of three popular software tools. For these evaluations, we used one million reads of sizes 16, 24 and 36 characters to be matched against the entire human genome (~3×10^9 characters). Our results show that our FPGA implementation can achieve up to 4X speed up over the fastest software tools.

The paper is organized as follows: the next section discusses the architecture of the naive method. The third section reports on the performance of the three software tools and compares them to the naive approach. The fourth section discusses related studies on string matching and the last section presents the conclusion and directions of future research.

II. NAÏVE IMPLEMENTATION

A. Architecture

As its name indicates, the naïve approach is a straightforward implementation that matches a subset of the reads against the streaming genome at every character position. The reads are initially loaded in registers before initiating the streaming of the genome. By adding up the number of matches in a given string (read), we can determine the number of mismatches.

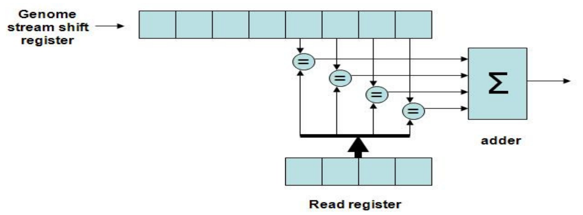

Figure 1: A read is held in the read register (here shown with four base pairs) for the duration of the current stream. The genome is streamed through the genome shift register.

Figure 1 illustrates the basic hardware architecture that carries out naive method string comparisons. In the set-up

1946-1488 2010
U.S. Government Work Not Protected by U.S. Copyright
DOI 10.1109/FPL.2010.78

phase, the read is loaded in the read register from the input stream. Subsequently, the genome is streamed through the genome shift register and matched against the read at every cycle. Note that the streams are of DNA characters and are 2-bits wide. To detect multiple mismatches, the results of the comparators are added using a pipelined saturating adder computing the total number of mismatches. For example, if we allow one or two mismatches then a saturating adder of two bits is used. If we allow three up to six mismatches then a saturating adder of three bits is needed. Figure 2 illustrates the diagram of the adder. However, for the case of searching perfect matches, instead of using saturating adders, an AND gate is used to save logic resources.

The basic structure of Figure 1 is replicated in two dimensions. Figure 3 shows the structure in Figure 1 replicated four times horizontally. The same stream is run through four string matching structures. We can also replicate the structure in Figure 1 vertically as shown in Figure 4. The idea is to split the genome into four streams, that are matched against the same read (string). So the same read is matched against multiple locations in the genome at the same time.

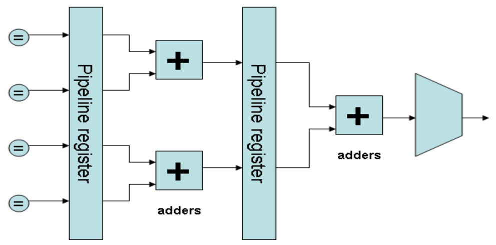

Figure 2: Diagram of pipelined saturating adder. The entire adder is organized as a pipelined tree to obtain a higher operating frequency.

The overall structure is shown in Figure 5. The match outputs of all the comparisons in one block are OR'ed together in order to reduce the pressure on the fan-out. The occurrence of a match, a rare event, is tagged as a location in each of the streams.

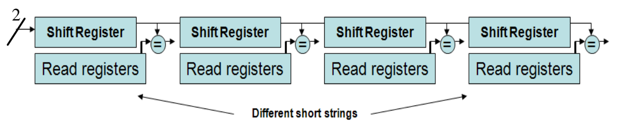

Figure 3: Chaining of stream shift registers.

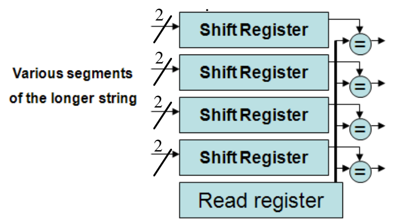

Figure 4: A block of four streams matched against one read register

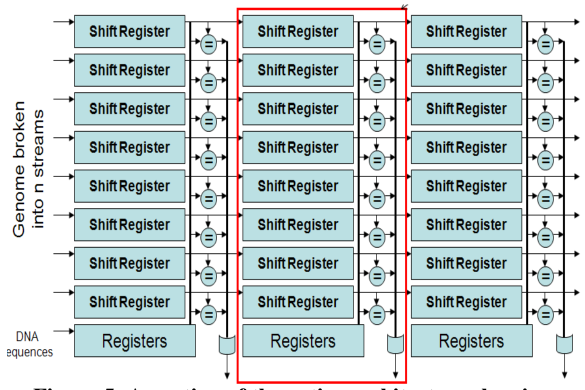

Figure 5: A portion of the entire architecture showing three chained block of eight shift registers each.

B. *Design Space Exploration*

This design space exploration identifies the structure that maximizes throughput which is defined by the number of character matches per second. It is therefore the product of the frequency, the number of string comparisons performed per cycle, and the read length.

The designs have are implemented on a Xilinx Virtex 5 LX330. The number of string comparisons per cycle aims at maximizing the resource utilization of the chip. We have implemented designs for reads with lengths of 16, 24 and 36, with one, two, four and eight streams and supporting three mismatches. All data are reported after physical synthesis, place and route. In all experiments, we attempted to maximize the FPGA area utilization as measured in number of slices.

The number of string comparisons per cycle on the FPGA is shown in Figure 6 and the frequencies in Figure 7. From these figures we can observe the following:

- The number of string comparisons on the FPGA increases as more streams are used. This observation can be explained using Figures 3 and 4. Both support four concurrent string comparisons, Figure 3 requires eight registers (four read registers and four shift registers) while Figure 4 requires five registers (one read register and four shift registers).
- Frequency generally decreases as more streams are used. This is because more streams incur fan-out penalties in comparing the read register to multiple shift registers. These penalties result to longer clock periods.

The throughput results are shown in Figure 8. We observe that using two or four streams results in the highest throughput for all read lengths. This is because having fewer streams results in higher frequencies. Although a single stream achieves the highest frequency, it utilizes more logic resources resulting in fewer character comparisons per cycle that fit on the FPGA. Using more streams utilizes less logic resources but operates at a lower frequency resulting in a lower throughput.

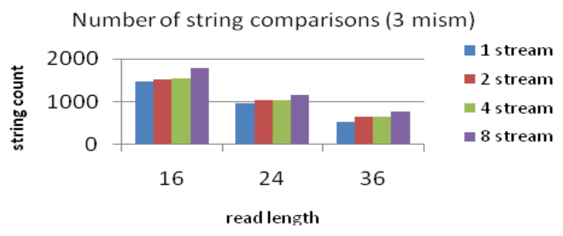

Figure 6: Number of string comparisons versus read length for 1 to 8 streams with ~90% of the slices of the FPGA (~60% register slices) using the naive implementation allowing three mismatches.

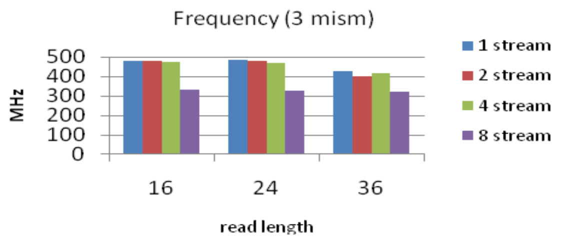

Figure 7: Operating frequency versus read length for 1 to 8 streams with ~90% of the slices of the FPGA (~60% register slices) using the naive implementation allowing three mismatches.

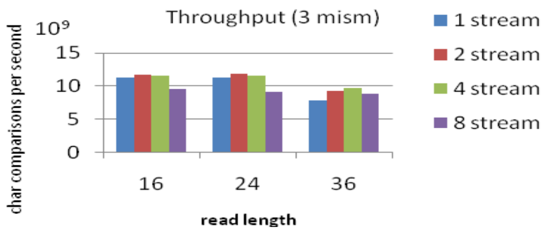

Figure 8: Throughput versus read length for 1 to 8 streams with ~90% of the slices of the FPGA (~60% register slices) using the naive implementation allowing three mismatches.

III. Comparison of Naïve Approach and Software

BLAST, which is the most commonly used tool in bioinformatics, was not included in this comparative evaluation because BLAST was designed to search a few reads in a very large database of DNA/protein data. Here the number of reads is in the millions, and the 'database' (i.e., reference genome) is only one. We compared the naive implementation to three popular software tools, namely MAQ [3], RMAP [6], and ELAND [2]. These tools use seeds to speed up the process of finding potential matches. Seeds are short substrings of fixed length on which a hash table is built. The indexes in the hash table are used to identify possible candidate positions. The candidate positions are then verified one by one. Table 2 shows the various programs used for this evaluation. Each software tool has limitations either in either the maximum read length or in the allowed number of mismatches.

The three tools were executed on a Quad-core Intel Xeon Harpertown at 2.5 GHz with 12 MB cache, but only one core was used in these experiments. This is to perform a chip to chip comparison between one core and one FPGA, as both can be parallelized. Note that the Harpertown is a 45 nm technology while the Xilinx Virtex 5 is a 65 nm technology device. We tested the tools using the human genome (3.3 billion characters) and one million reads of 16, 24 and 36 characters in length while allowing zero, two and three mismatches.

Table 1: Software tools possible configurations

Program	Read Length	Allowable Mismatches
MAQ	<=63	3
RMAP	16-64	Up to 64
ELAND	16-32	2

Table 3 shows the various execution times of the tools evaluated for the three read lengths and three allowed mismatches. Table 4 shows the execution time of the fastest software tool with the execution time of the FPGA based on throughput. The rightmost column shows the speed up of the FPGA implementation. Observe that it achieves speed-ups as high as 4X over the software tool. However, ELAND achieves better execution time than the FPGA for two mismatches and read length of 24, but has a worse execution time than the FPGA for read length of 16. Furthermore, ELAND can only allow up to two mismatches and can only match reads of size up to 32 characters. In general, FPGA speed up generally decreases for increasing read lengths. This is mainly because when reads are longer, fewer reads can be placed on the FPGA thereby decreasing the throughput of the FPGA.

Table 2: Execution time of software tools for different read lengths and allowing different count of mismatches.

		Execution time (s)		
Mismatch count	Read length	16	24	36
3	RMAP	11,531	16,662	19,931
	MAQ	285,97	61,117	32,791
	ELAND	n/a	n/a	n/a
2	RMAP	16,936	19,182	19,477
	MAQ	95,727	25,595	19,947
	ELAND	11,463	3,450	n/a
0	RMAP	11,602	16,106	17,704
	MAQ	n/a	n/a	n/a
	ELAND	n/a	n/a	n/a

IV. Related Work

The problem of string matching on FPGAs has been extensively studied and researched with focus on two application domains: network intrusion detection systems (NIDS) and bioinformatics. In intrusion detection, packets are scanned for the presence of signatures of known network attacks. In bioinformatics, DNA or amino acid sequences are searched on a reference genome allowing a limited number of character mismatches during matching.

Table 3: Best software tool and FPGA execution time.

Mis-match	Read length-Program	Software time (s)	FPGA time (s)	Speed up
3	16-RMAP	11,531	4,510	**2.56**
	24-RMAP	16,662	6,700	**2.49**
	36-RMAP	19,931	12,300	**1.62**
2	16-ELAND	11,463	5,720	**2.00**
	24-ELAND	3,450	8,200	**0.42**
	36-RMAP	19,477	13,300	**2.36**
0	16-RMAP	11,602	2,820	**4.11**
	24-RMAP	16,106	5,020	**3.21**
	36-RMAP	17,074	7,230	**2.36**

One of the major differences in requirements between NIDS and bioinformatics is allowing character mismatches which is known as approximate string matching. Mismatches are necessary to account for errors from sequencing machines and mutation in the genome itself.

An implementation of approximate string matching is related to the adaptation in hardware of a software tool known as BLAST [14]. The seed generation phase of the BLAST heuristic is implemented in hardware in [9]. A more recent study explores the design space of the BLAST hardware implementations [10]. This study includes tuning memory elements of the architecture such as registers and FIFO queues.

A major direction in approximate string-matching in bioinformatics utilizes various dynamic programming algorithms to compute the edit distance. Edit distance is the number of character conversions to transform one string to the other. The two main algorithms in focus are Needleman-Wunch and Smith-Waterman. A study implements a hardware platform that can be parameterized for these two algorithms [11]. The parameters include length of pattern, number of symbols, and allowed mismatches. Other studies concerned on dynamic programming focused on generating systolic arrays on FPGAs of the Smith-Waterman algorithm [8, 11, 12, 13]. Another dynamic programming approach [15] also involved in computing the edit distance of two strings besides Needleman-Wunch and Smith-Waterman is implemented in [4].

Another option to perform string matching is to break the text into various sections [5]. The text sections are streamed and compared in parallel to the patterns. This is the basis of the naive method discussed in this paper.

V. Conclusion

In this paper, we investigated the feasibility of a hardware approach using naive method for large scale string matching. Surprisingly, the naive method achieves higher throughput than the convolution-based approach, when implemented in hardware. The naive method is also faster than existing software tools, showing a 1.6X-4X speed up. The limitation of our hardware approach is related to longer read lengths. For longer read lengths, only few reads can be placed on the FPGA, which increases the execution time because a higher number of streaming of the genome through the FPGA is needed. The trend of new sequencing machines is to output longer and longer reads at each technology advance, which would require different hardware architecture.

References

[1] Illumina Inc. http://www.illumina.com

[2] O. Cret, Z. Mathe, P. Ciobanu, S. Marginean, and A. Darabant. *A Hardware Algorithm for the Exact Subsequence Matching Problem in DNA Strings*. Romanian Journal of Information Science and Technology, 2009.

[3] H. Li, J. Ruan, and R. Durbin. *Mapping Short DNA Sequencing Reads and Calling Variants Using Mapping Quality Scores.* Genome Research, Vol. 18, No. 11, pp1851-1858, 2008.

[4] S. Mikami, Y. Kawanaka, S. Wakabayashi, and S. Nagayama. *Efficient FPGA-based Hardware Algorithms for Approximate String Matching*. In the Proceedings of International Technical Conference on Circuits/Systems, Computers and Communications, 2008.

[5] A.N.M.E. Rafiq, F. Gebali, and M.W. El-Kharashi. *A Systolic Array Structure for String Searching*. In the Proceedings of the International Conference on Electrical, Electronic and Computer Engineering, 2004.

[6] A.D. Smith, Z. Xuan, and M.Q. Zhang. *Using quality scores and longer reads improves accuracy of Solexa Read Mapping*. BMC Bioinformatics , February, Vol. 9, No. 128, pp1471-2105, 2008.

[7] Applied Biosystems. http://www3.appliedbiosystems.com/AB_Home/applicationstechnologies/SOLiDSystemSequencing/index.htm

[8] B. Buyukkurt and W. Najjar. *Compiler Generated Systolic Arrays for Wavefront Algorithm Acceleration on FPGAs*. In the Proceedings of 18th International Conference on Field Programmable Logic and Applications (FPL), 2008.

[9] A. Jacob, J. Lancaster, J. Buhler, and R. Chamberlain. *FPGA accelerated seed generation in Mercury BLASTP*. International Symposium on Field Programmable Custom Computing Machines (FCCM), 2007.

[10] E. Sotoriades and A. Dollas. *Design Space Exploration for the BLAST Algorithm Implementation.* International Symposium on Field Programmable Custom Computing Machines (FCCM), 2007.

[11] G. Caffarena, S. Bojanic, J. Lopez, C. Pedreira, and O. Nieto-Taladriz. *High-Speed Systolic Array for Gene Matching.* International Symposium on Field Programmable Gate Arrays (FPGA), 2004.

[12] M. Gok and C. Yilmaz, *Efficient Cell Designs for Systolic Smith-Waterman Implementations.* International Conference on Field Programmable Logic and Applications (FPL), 2006.

[13] P. Zhang, G. Tan, and G. Gao. *Implementation of the Smith-Waterman algorithm on a Reconfigurable Supercomputing Platform.* Conference on High Performance Networking and Computing, 2007.

[14] BLAST. http://blast.ncbi.nlm.nih.gov/Blast.cgi

[15] R.A. Wagner and M.J. Fisher. *The string-to-string correction problem.* Journal of ACM, vol 21, pp. 168-173. 1974.