Reconfigurable Acceleration of Short Read Mapping

James Arram*, Kuen Hung Tsoi*, Wayne Luk* and Peiyong Jiang†
*Department of Computing, Imperial College London
†Department of Chemical Pathology, The Chinese University of Hong Kong
{jma11, khtsoi, wl}@imperial.ac.uk

Abstract—**Recent improvements in the throughput of next-generation DNA sequencing machines poses a great computational challenge in analysing the massive quantities of data produced. This paper proposes a novel approach, based on reconfigurable computing technology, for accelerating short read mapping, where the positions of millions of short reads are located relative to a known reference sequence. Our approach consists of two key components: an exact string matcher for the bulk of the alignment process, and an approximate string matcher for the remaining cases. We characterise interesting regions of the design space, including homogeneous, heterogeneous and run-time reconfigurable designs and provide back of envelope estimations of the corresponding performance. We show that a particular implementation of this architecture targeting a single FPGA can be up to 293 times faster than BWA on an Intel X5650 CPU, and 134 times faster than SOAP3 on an NVIDIA GTX 580 GPU.**

I. Introduction

DNA contains a long sequence of pairs of nucleotide bases which can be abstracted into a character string with an alphabet $\Sigma = \{\text{'A'}, \text{'C'}, \text{'G'}, \text{'T'}\}$. DNA sequencing is the process of identifying the order of the nucleotide bases in a DNA molecule. This process has been utilised in a wide range of applications; for example in medicine, analysis of a patients genetic information can be used in diagnosing hereditary diseases. Next-generation sequencing (NGS) machines are able to rapidly and inexpensively produce sequenced data. To improve the throughput and measurement accuracy of these machines, shorter sequences are processed, allowing tens of billions of bases to be sequenced per day. These short sequences can be created by cutting the DNA strand randomly, producing millions of short fragments. As a consequence of this action, the position and orientation information of the fragments with respect to the DNA is lost. Based on the assumption that DNA sequences within a species are similar, the sample DNA can be reconstructed by determining the location of the short fragments (*short reads*) in a known reference genome of the species. An aligner system is used to find the possible positions of these short reads in the reference DNA.

The performance of NGS machines has been improving at a rate faster than Moore's law for almost a decade. Large computer clusters are often used to process short read data generated by a single sequencing machine. However, the processing speed of computer clusters does not seem to grow as fast as the speed of sequencing machines and as a result the performance of a software based aligner is usually the bottleneck of a bioinformatic analysis flow.

FPGA technology is a promising candidate for accelerating this application, which involves highly-parallel bit-oriented operations. It is found that statistically around 70–80% of the short reads in a typical workload can be exactly aligned to a reference sequence. Some algorithms, such as FM-index [1], are particularly efficient at exact matching. Based on these observations, we propose a new reconfigurable heterogeneous design containing two kinds of string matchers: exact string matchers (ESMs) and approximate string matchers (ASMs). The performance of the aligner design is benefited from the decoupling of the ESM and ASM as a) the matchers can be highly optimised for the expected input data type, resulting in shorter aligning time and smaller circuit size and b) the decoupling improves the flexibility of optimising the population of each type of string matcher according to the intended workflow, this results in a higher achievable level of parallelism.

The major contributions of this work include:

- A two-stage architecture for accelerating short read sequence alignment based on the decoupling of the ESM and ASM. We characterise interesting regions of the design space, including homogeneous, heterogeneous and run-time reconfigurable designs and provide estimations of the corresponding performance (Section III).
- A hardware implementation of the reconfigurable architecture targeting a single FPGA. The ESM is based on FM-index and the ASM based on a seed and expansion strategy. Various features of the string matcher designs, such as methods to reduce the memory size and improve the throughput are covered (Section IV).
- Performance evaluation of the hardware implementation, together with comparisons against some of the fastest software solutions on multi-core processors, GPUs and hardware solutions on FPGAs (Section V).

II. Background and Related Work

A. Exact String Matcher

There exist a number of well developed methods for efficient exact pattern matching. For example, in [2] a direct comparison method is used in which the bases from a streaming reference sequence and a stationary short read are compared. Our approach for exact alignment is based on FM-index [1], a data structure that has inspired several software tools for analysing genetic sequences such as Bowtie [3] and SOAP2 [4]. This index combines the properties of suffix array with the Burrows-Wheeler transform (BWT) [5], to provide an efficient method for finding all occurrences of a pattern Q within a long reference sequence R.

To compute the BWT of a reference sequence R, first R is terminated with a unique character '\$', which has the smallest lexicographical value. All rotations of R are generated and sorted lexicographically, then the BWT sequence is extracted from the last column of the sorted rotations. Table I(a) illustrates an example of generating the BWT for the reference sequence $R =$ ACTAGCTA. The character strings preceding the '\$' symbol in the sorted rotations column form a suffix array (SA), which indicates the starting position of each possible suffix in R.

Table I

(a) BWT GENERATION AND SUFFIX ARRAY REPRESENTATION OF REFERENCE SEQUENCE R. **(b)** VALUES OF FUNCTIONS $c(x)$ AND $s(x, i)$.

(a)

R = ACTAGCTA		
Index	SA	Rotations
0	8	\$ACTAGCTA
1	7	**A**\$ACTAGCT
2	3	**AGCTA**\$ACT
3	0	**ACTAGCTA**\$
4	5	**CTA**\$ACTAG
5	1	**CTAGCTA**\$A
6	4	**GCTA**\$ACTA
7	6	**TA**\$ACTAGC
8	2	**TAGCTA**\$AC
$BWT(R)$ = ATT\$GAACC		

(b)

$s(x, i)$

i	A	C	G	T
0	1	0	0	0
1	1	0	0	1
2	1	0	0	2
3	1	0	0	2
4	1	0	1	2
5	2	0	1	2
6	3	0	1	2
7	3	1	1	2
8	3	2	1	2
$c(x)$	{0	3	5	6}

After generating the BWT sequence and the suffix array representation of the reference sequence, the functions $c(x)$ and $s(x, i)$ are defined. $c(x)$ (the frequency) is the number of symbols in the reference sequence that are lexicographically smaller than x, and $s(x, i)$ (the occurrence) is the number of occurrences of the symbol x in the BWT sequence from the 0^{th} position to the i^{th} position. These functions are usually implemented as lookup tables using an array structure. Table I(b) illustrates the $c(x)$ and $s(x, i)$ functions for the reference sequence R.

The Suffix Array (SA) interval of a pattern Q is defined as $[k, l]$. The pointers k and l are respectively the smallest and largest indices in the suffix array which starts with Q. To search for a pattern Q within the reference sequence, k and l are initialised to the first and last indices of the suffix array table respectively. Using equations 1 and 2 the SA interval is updated for each character in Q, moving from the last character to the first.

$$k_{new} = c(x) + s(x, k_{current} - 1) + 1 \qquad (1)$$

$$l_{new} = c(x) + s(x, l_{current}) \qquad (2)$$

Figure 1 shows an example of searching the pattern Q = CT in the reference sequence $R =$ ACTAGCTA. First, k and l are initialised to 0 and 8 respectively. Then equations 1 and 2 are applied twice, corresponding to the number of characters in Q.

1^{st} iteration: x = T	2^{nd} iteration: x = C
k_{new} = c(T) + s(T, 0) + 1	k_{new} = c(C) + s(C, 6) + 1
= 6 + 0 + 1 = 7	= 3 + 0 + 1 = 4
l_{new} = c(T) + s(T, 8)	l_{new} = c(C) + s(C, 8)
= 2 + 6 = 8	= 3 + 2 = 5

Figure 1. Example of searching using the FM-index

After the second iteration, k and l become 4 and 5 respectively. Since $k \leq l$, the pattern can be found in the reference sequence. Note that if $k > l$ for an iteration, the pattern cannot be exactly matched to the reference sequence. The suffix array elements corresponding to each index within the SA interval give the location of the pattern in the reference sequence. Table I(a) indicates that index 4 and 5 map to positions 5 and 1 respectively in the reference sequence.

B. Approximate String Matcher

There exist a number of efficient and accurate methods for approximate pattern matching. For example in [6] a bi-directional BWT is developed which supports the detection of mismatches and indels. Our approach for approximate alignment is based on a *Seed and expansion* strategy, which extracts seeds (even shorter sequences) from the short read and uses these seeds to find the possible matching locations. The matching locations can be found by lookup in a hash table [7] or using suffix tree based algorithms, such as the FM-index. Seeds can be generated as every possible fixed length subsequence (e.g. BLAST [7]), or by partitioning the short reads into fixed length segments (e.g. Bowtie [3]). The short read is then aligned to the reference genome at each matching location using the Smith-Wateman algorithm [8].

In the Smith-Waterman algorithm, a scoring matrix is used to perform local sequence alignment, where a short read is aligned to a local section of the reference genome. For the sequences S and T where $|S| = n$ and $|T| = m$, the scoring matrix H is constructed using equations 3 and 4.

$$\text{Initialisation} \begin{cases} H(i, 0) = 0 & 0 \leq i \leq n \\ H(0, j) = 0 & 0 \leq j \leq m \end{cases} \qquad (3)$$

$$H(i, j) = \max \begin{cases} 0 \\ \begin{cases} H(i-1, j-1) + \sigma(\text{match}) & \text{if } S_i = T_j \\ H(i-1, j-1) + \sigma(\text{mismatch}) & \text{if } S_i \neq T_j \end{cases} \\ H(i-1, j) + \sigma(\text{deletion}) \\ H(i, j-1) + \sigma(\text{insertion}) \end{cases}$$

$$1 \leq i \leq m, 1 \leq j \leq n \qquad (4)$$

Here $\sigma(x)$ is the score penalty for x, which is determined by the gap-scoring scheme. The scores can be adjusted for different comparison requirements. Table II illustrates the construction of the scoring matrix for the alignment of a pattern Q = CT and a reference R = ACTAGCTA.

Table II
EXAMPLE OF CONSTRUCTING THE SCORING MATRIX FOR THE PATTERN Q = CT AND REFERENCE SEQUENCE R = ACTAGCTA.

	-	A	C	T	A	G	C	T	A
-	0	0	0	0	0	0	0	0	0
C	0	0	2	1	0	0	2	1	0
T	0	0	1	4	3	2	1	4	3

$\sigma(\text{match}) = +2 \quad \sigma(\text{mismatch}) = -1$
$\sigma(\text{deletion}) = -1 \quad \sigma(\text{insertion}) = -1$

The highest score in the scoring matrix is used to determine whether the pattern can be mapped to the reference sequence within an allowed diversity. In Table II the highest score is 4 which indicates that the pattern can be exactly mapped to the reference sequence. A traceback step can be included which reconstructs a string representation of the optimal alignment.

C. Related Work

There are several hardware accelerators for genetic sequence analysis. In [9], an FM-index based algorithm is proposed for FPGA. It concludes that using a single large table for FM-index is more area efficient than splitting into multiple smaller tables. The performance is 1000 reads in 60.2 μs if mismatches are not allowed. In [2], a short read mapper is developed using a direct comparison approach. A single LUT is used to compare 2 bases from a streaming reference sequence and a stationary short read. Using a Xilinx XC6V-LX550T FPGA, the system achieves 500000 reads in 212 seconds. In [10], the short read alignment is performed by CPU and FPGA collaboratively. The indexing part is performed by CPU and then the short reads, as well as the corresponding reference segments, are sent to the FPGA for pairwise matching. The design achieves around 16 million reads in 110 seconds using a Xilinx XC5V-LX330 FPGA. In [11], an FPGA based short read alignment accelerator is proposed based on indexing of the reference sequence, with Smith-Waterman alignment performed in FPGA. The main optimisation in this work is to reduce the size of the candidate alignment location (CAL) lookup table. The system, and also the CAL table, is partitioned into 8 Pico M-503 boards each with one XC6V-LX240T FPGA. This 8-FPGA system can map 50 million short reads in 34 seconds. In [12], a backtracking version of the FM-index is proposed for FPGA. A bi-directional search is used to improve the performance of approximate matching. The design achieves 1 million reads in 2.6 seconds for up to two mismatches. Our work differs from previous hardware designs by decoupling the ESM and ASM, allowing optimisation of the population of each matcher according to the intended workflow.

III. ACCELERATOR SYSTEM ARCHITECTURE

In a software aligner, the short reads enter the approximate alignment process once the exact alignment fails. No additional CPU cycles are wasted for changing the alignment algorithm. This is not applicable to specialised accelerators since the alignment process is fixed in the hardware circuit. To avoid wasting cycles when changing the alignment algorithm, most specialised accelerator designs utilise a generic alignment circuit. However, in this approach the circuit for the approximate alignment algorithm permanently consumes hardware space and power. Since approximately 70-80% of short reads can be exactly mapped in a typical workload, this approach is suboptimal.

To address this problem, we propose an architecture incorporating specialised processors for exact and approximate alignment, allowing the population of each type of matcher to be optimised according to the intended workflow. When a short read is to be aligned, it is first sent to the ESM. The ESM is designed to be compact and efficient for exact alignment. If the short read fails to be aligned by the ESM, it is subsequently processed by the ASM for more complicated analysis. The overall performance of the aligner is benefited from the decoupling between the exact and approximate string matchers as a) the matchers can be highly optimised for the expected input data type, this results in shorter aligning time and smaller circuit size and b) the decoupling improves the flexibility of optimising the population of each type of matchers according to the intended workload, resulting in higher achievable parallelism.

Due to the decoupling of the ESM and ASM there is a multitude of ways to configure the alignment accelerator. Here we characterise the interesting regions of the design space, including homogeneous, heterogeneous and run-time reconfigurable designs and provide back of envelope estimations for the corresponding performance. Furthermore, we analyse the conditions in which each design is the most favourable. The key parameters used in the analysis are defined in Table III.

Table III
KEY SYSTEM PARAMETERS.

N	number of short reads
α	percentage of approximately matched short reads
N_E, N_A, N_S	number of ESMs, ASMs and software threads
T_E, T_A, T_S	run time for an ESM, ASM and a software thread to match a short read
R_E, R_A, R_P	resources of an ESM, ASM, and an accelerator platform
R_O	resources for overhead such as memory controllers
T	alignment time

D1: statically populated ESM, software approximate matching

In this design the accelerator is populated with only ESMs and the unaligned short reads reported by the accelerator are processed by software. The number of ESMs is limited by the amount of critical resources of the ESM in the accelerator. The effective parallelism is also limited by the number of memory channels, since the memory accesses from the ESMs must not interfere with each other. The maximum number of ESMs, N_E, is given by:

$$N_E = (R_P - R_O)/R_E \tag{5}$$

Note that R_O can include FPGA place and route overhead between processors. The worst case alignment time is:

$$T = \frac{N}{N_E}T_E + \frac{\alpha N}{N_S}T_S \tag{6}$$

High performance can be achieved when the software aligner is able to process the unaligned short reads as soon as they are received, maximising the parallelism of the design.

This design should be considered when detailed alignment output is required. The reason being that hardware approximate alignment designs often compromise the functionality of the aligner in order to improve the performance.

D2: statically populated ESM and ASM

In this design, both the ESMs and the ASMs are instantiated in the accelerator. All short reads are processed by the ESMs first. The short reads which cannot be exactly matched by the ESMs are internally forwarded to the ASMs for approximate matching.

Without any platform specific constraints, the population of these two matchers should agree with the ratio of exact matches to approximate matches in the short read data. However, in practice the number of matchers is limited by the available resources in the accelerator. The population ratio should fulfill:

$$N_E R_E + N_A R_A + R_O \leq R_P \tag{7}$$

Once these short reads are forwarded, the ESMs continue to process new short reads in the data set. This introduces an overlapping period when both the ESMs and the ASMs are processing short read data. In the worst case, all approximate matches are at the end of the data set, such that the overlap period is minimised. This worst case alignment time is:

$$T = \frac{N}{N_E}T_E + \frac{\alpha N}{N_A}T_A \tag{8}$$

High performance can be achieved when the approximate matches are distributed uniformly in the workload, maximising the overlap period.

This design should be considered for small workloads where the reconfiguration time of the device is larger than the total alignment time.

D3: dynamically reconfigured ESM and ASM

In this design, we take advantage of the reconfigurability of the accelerator device. We consider dynamic reconfiguration since a) a static design will need to contain at least one ASM which is much larger than the ESM. Given that approximately 70%-80% of the short reads can be dealt with by the ESM, it would make sense to populate the accelerator first with as many ESMs as possible, then reconfigure it with ASMs to process the unaligned short reads and b) For a typical workload, the reconfiguration time is small compared to the alignment time, therefore dynamic reconfiguration does not bottleneck the design. Furthermore if components of the ESM are used in the ASM, a process such as partial reconfiguration could be used to efficiently swap in the ASM.

Initially, the accelerator is fully populated with ESMs. The short reads are streamed to the accelerator and are processed by the ESMs first. The short reads which are unable to be aligned by the ESMs are stored in on-board memory. Once all short reads are processed by the ESMs, the accelerator device is reconfigured to become fully populated with ASMs. The previously stored unaligned short reads are then loaded and processed by the ASMs. This design allows all the short reads to be processed within the accelerator using specialised matchers which are optimised for the data characteristics. The overhead of this design is the additional memory storage for the unaligned short reads, and the reconfiguration time. Note that it is possible to process the ESM results in the host program and stream the unaligned short reads to the accelerator after reconfiguration. In this case no additional on-chip memory is required to store the unaligned short reads, however the performance is limited by the PCIe bandwidth.

Since the accelerator is fully populated by either type of the matchers, the maximum number of ESMs is the same as D1 from Equation 5 and the maximum number of ASMs is:

$$N_A = (R_P - R_O)/R_A \tag{9}$$

The total alignment time is:

$$T = \frac{N}{N_E}T_E + \frac{\alpha N}{N_A}T_A + T_{reconfig} \tag{10}$$

where $T_{reconfig}$ is the time for reconfiguring the accelerator device. Here we assume that the external memory of the accelerator is large enough to store all the unaligned data. If memory size is a limiting factor, then the process must be repeated multiple times which depends on the unaligned short read size and the available memory for data buffering. As a result, the reconfiguration overhead in Equation 10 has to be multiplied accordingly.

This design should be considered for typical short read alignment workloads where millions of short reads are aligned to a reference sequence. In such workloads the design benefits from maximising the population of each type of matcher on the accelerator device. For most devices, the reconfiguration time is much smaller than the alignment time, therefore the overhead maximising the population of each type of matcher is negligible (even with multiple reconfigurations). This design can be extended by replacing the ASM with specialised approximate string matchers (SASMs), each capable of efficiently testing a specific diversity (e.g. one mismatch). Reconfiguration can be used to maximise the population of each SASM according to the diversity being tested for, further improving the design performance.

IV. Architecture Implementation

Here we present a particular hardware implementation of the architecture presented in Section III, targeting a single FPGA device. The architecture is completely general, therefore different ESM and ASM designs from the ones used here are possible.

In our implementation, the ESM uses the FM-index to align the short reads to the reference sequence. We choose the FM-index as it is the most space and time efficient data structure for exact matching. This allows a large number of ESMs to populate the FPGA, resulting in a higher achievable level of parallelism. Since the ESM is the critical component in the proposed architecture, various methods are applied to improve the performance. As shown in Section IV-A, the methods are specific to the underlying FPGA platform to achieve a throughput of one aligned base per cycle. In the proposed architecture, multiple ESMs work independently in a single accelerator.

The ASM uses a seed-and-expansion strategy to align more diverse sequences. The input short reads are segmented into fixed length seeds and the FM-index is used to align the seeds to a large number of exactly matching locations in the reference sequence. After finding out the most probable alignment locations, the similarity is then evaluated using the Smith-Waterman algorithm. The Smith-Waterman algorithm is chosen as it is guaranteed to find the optimal alignment according to the chosen gap scoring scheme.

Figure 2 illustrates the system level architecture of our implementation. Auxiliary data structures, such as the BWT sequence, the occurrence array and the suffix array, are transferred from the host computer to the FPGA accelerator in advance. This procedure has negligible impact since it is performed only once for aligning hundreds of millions of short reads. These data are static for read only accesses throughout the alignment process. Once the FPGA platform is initialized, the host computer starts streaming the short reads to the accelerator. All short reads will be first processed by the ESMs where alignment results are streamed back to the host system when aligned exactly. If no exact alignment position is found by the ESM, the short reads are processed by the ASM, where a longer time is taken for running the approximate alignment algorithm. A controller unit is developed to coordinate the interactions between host computer and the alignment processors.

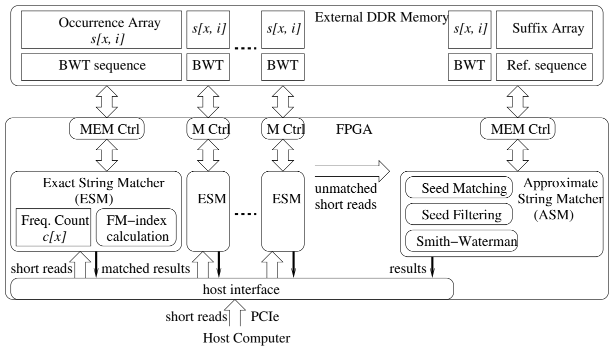

Figure 2. System architecture overview.

For exact alignment, all the alignment locations of the short reads are reported to the host program. For approximate alignment, the cost representing the similarity is reported, as well as the alignment locations. No traceback step is performed by the the ASM, therefore the string representation of the optimal alignment must be reproduced by the CPU, adding an overhead.

A. ESM: Exact String Matcher

The ESM provides a hardware version of the FM-index algorithm described in Section II-A. Our design extends the one in [9] by allowing for alignment against the full human reference genome (3.2G bases). This requires that memory external to the FPGA device is used to store the FM-index data structures, rather than on-chip BRAMs. Figure 3 illustrates the architecture of our ESM implementation.

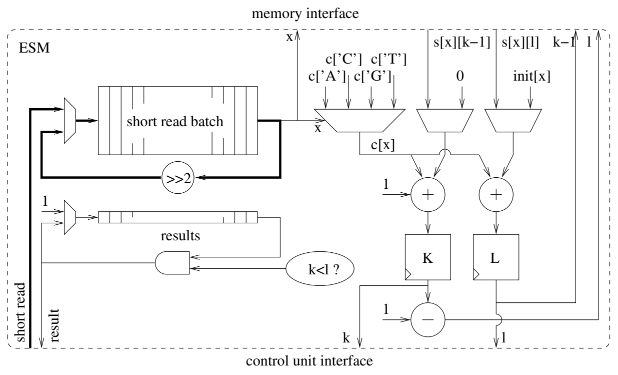

Figure 3. ESM architecture.

The ESM design has the following features which improve the efficiency:

• A scheme to reduce the memory footprint of the occurrence array, $s[x][i]$, so that it can fit on most FPGA platforms and allows multiple copies of this array to be associated with multiple ESMs. This is achieved by storing a full range marker for every d elements in the occurrence array. To reconstruct the occurrence value of $s[x][i]$, the following components are summed: a) the lower marker value relative to position i and b) the result from counting the occurrence of symbol x in the BWT sequence between the lower marker position and i. In this approach we store the marker values and the BWT sequence. The required memory storage is then signifcantly reduced to:

$$4 \times \frac{3.2G \times 32bit}{d} + 3.2G \times 3\ bits$$

For example, the memory footprint is reduced from 51.2G bytes to 2G bytes for d = 64.

• A method to maximise the throughput of the ESM, which negates the latency of external memory access. This is achieved by interleaving the processing of multiple short reads. In this approach, the ESM contains a buffer able to store a small batch of short reads. In each clock cycle a symbol from a different short read is selected and the corresponding external memory is requested. This allows the processing of short reads while others are waiting for external memory. As a result the design is fully pipelined with a throughput of one aligned base per cycle. This enables the equivalent of a multi- threaded program aligning multiple short reads in parallel, but with zero thread switching overhead.

• A scheme to reduce the number of connections to the external memory and make effcient use of the available memory bandwidth. This is achieved by interleaving the marker array and BWT sequence such that the occurrence array markers and the corresponding BWT sequence segments are grouped together in external memory. By interleaving, both the relevant marker and corresponding BWT sequence segments can be accessed in a single memory access, reducing the memory access frequency and external memory connections by 50%.

B. ASM: Approximate String Matcher

The ASM provides a hardware version of the seed and expansion based strategy described in Section II-B. Our design draws inspiration from the work in [11], although we use the FM-index described in Section IV-A to align the seeds, rather than a hash table based approach. Furthermore, we perform the Smith Waterman alignment on the most frequent candidate alignment locations (CALs) rather than on all unique CALs. Figure 4 illustrates the basic architecture of our ASM implementation.

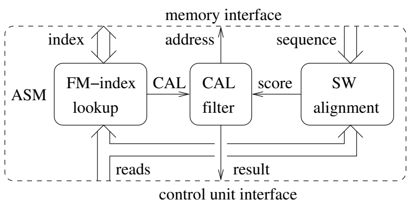

Figure 4. ASM architecture.

In the ASM, a short read is partitioned into fixed length seeds on which exact matches are performed. The length of seeds, and thus the number of seeds, is configurable by user application at runtime. By controlling this parameter, we can control the permitted number of mismatches and gaps in the alignment process. The FM-index is used to map the seeds to the reference sequence as a) it is compact such that multiple independant exact matches can be performed and b) the time complexity of finding *all* matching locations is linear in length of the seed and independent of the length of the reference sequence. The resulting locations at which the seeds can be exactly mapped are defined as the candidate alignment locations. The CALs for each seed are stored in a table on the FPGA using BRAMs as storage units. The most frequently occurring CALs are then used as the address to retrieve a segment of the reference sequence. By selecting the most frequent CALs, no cycles are wasted on processing CALs which are unlikely to align to the reference sequence within the permitted diversity. The size of the reference sequence segment is a few bases larger than the short read. The actual starting position of segment is set to be a few bases ahead of the CAL. These parameters usually depend on the maximum allowed gap size. The similarity score between the short read and the sequence segment is computed using the Smith-Waterman algorithm as described in Section II-B. The Smith-Waterman algorithm is performed in a systolic array such that each base in the short read has its own systolic element. The maximum score in the scoring matrix is used to determine if the short read is mapped to the CAL within the allowed diversity.

V. Performance Evaluation

In this work, we use the Human Genome version 18 reference sequence [13] and short read data sampled directly from the reference sequence. We insert noise in random positions in the short reads to simulate mismatches between the short read and reference sequence.

Since memory size is the major concern in this application, we target FPGA platforms such as the Maxeler MAX3 FPGA boards. In the MAX3 platform, the Xilinx Virtex-6 SX475T FPGA is associated with 24G bytes of external DRAM. The system can also support up to 15 independent memory controllers. In this platform a design is described using the MaxJ language, an extension of the Java language. The MaxCompiler then maps the design into an FPGA implementation and enables its use from a host application.

The ESM design is implemented in the Maxeler MaxJ language and in Verilog. Both versions implement the optimisations discussed in Section IV-A. Table V shows the resource usage for the ESM implemented in the Maxeler MaxJ language and in Verilog. Both implementations support up to 100 bases for a single short read query and contain a 56-entry short read buffer to allow for the processing of multiple short reads as discussed in Section IV-A. The occurrence

Table IV
PERFORMANCE COMPARISON.

	read size (base)	ref. size (base)	read count (million)	platform	clock freq. (MHz)	device(s)	core(s)	run time (s)	bps (million)
SOAP2 [4]	90	3.1G	98.2	Intel Xeon E5335	2000	1	4	5547	1.59
SOAP3 [14]	100	3.1G	70.7	NVIDIA GTX 580	900	1	512	1839	3.84
BWA [15]	76	3.1G	82	Intel X5650	2670	1	20	3540	1.76
Bowtie [3]	35	3.2G	82	Intel X5650	2670	1	20	2760	1.04
this work D2	90	3.1G	82	Xilinx Virtex-6 SX475T	200	1	7	3454	**2.13**
Design in [11]	76	3.1G	50	Xilinx Virtex-6 LX240T	250	8	8	34	112
Design in [16]	101	107M	18	Convey HC-1	150	4	1	138	59.2
Design in [17]	76	4.9M	1	Xilinx Virtex-5 LX330	-	1	4	21	3.6
this work D3	90	3.1G	82	Xilinx Virtex-6 SX475T	200	1	6	49	**150**
this work D4	90	3.1G	82	Xilinx Virtex-6 SX475T	200	1	5	14.3	**516**

array is stored in external memory with the marker values for every 64 bases. As a result the total storage requirement for a single ESM is 2G bytes. The highest clock speed of this design is found to be 200MHz and is achieved in the Verilog implementation.

Table V
FPGA RESOURCE UTILISATION REPORT.

	LUT	D Flip-Flop	freq. (MHz)
MaxCompiler ESM	3486	10143	150
Verilog ESM	570	671	200
Verilog ASM	1154	1790	200

In our implementations, the number of string matchers populating the FPGA is limited by the BRAM resources and the number of memory controllers supported by the device. In the current designs each ESM is associated with two memory controllers for accessing the k and l values concurrently, whilst the ASM requires two additional memory controllers for accessing the reference sequence and the suffix array.

With no limit on the number of memory controllers, we estimate that each FPGA would be able to support nearly 100 of each type of string matcher, given that the Verilog implementations consume fewer than 1% of the available hardware resources. Table V indicates that the Maxeler implementation uses more FPGA resources than the Verilog implementation; this is due to platform wrapper overheads.

For the ASM architecture, we limit the length of a seed to 20 bases in order to reduce the size of the CAL table and the resources required to select the most frequently occurring CALs. We set the threshold of the Smith-Waterman alignment according to the permitted diversity at runtime. Any short read which scores less than this threshold is considered to be unaligned. Table V shows the resource usage for the ASM implemented in Verilog. The highest clock speed of this design is found to be 200MHz.

Since different packages report their performance using different data sets and different designs, it is difficult to directly compare using the raw results. To better assess the performance of various designs, we define the bases per second (*bps*) value as a normalised performance merit.

$$bps = \text{read size} \times \text{read count}/\text{process time}$$

In Table IV we compare the designs in Section III with existing software and hardware aligners. We use the Verilog ESM and ASM in our performance estimations. The following commands are used to run the software aligners:

- ./bowtie -v 5 -S -p 20 reads.fq temp.sam
- ./soap -D Hg18.fa.index -a reads.fq -l 15 -v 5 -p 20 -m 0 -x 600 -r 1 -2 soap2.out1 -o soap2.out2
- ./bwa aln -l 12 -k 3 -n 5 -t 20 reads.fq > temp.sai

D1 has 7 ESMs and uses 14 out of the 15 available memory controllers. SOAP2 is used in D1 as the software approximate aligner, and the software runs 20 threads on a server with dual Intel Xeon X5650 CPUs at 2.67GHz. Due to the large number of memory controllers required by the ASM, D2 has 5 ESMs and 1 ASM. D3 has 7 ESMs in the exact alignment phase and 3 ASMs in the approximate alignment phase. The static memory structures in D3, such as the BWT sequence and the suffix array, consume about 22.4GB external memory. These data are shared between the ESMs and the ASMs to reduce data loading overhead. For D3, the ESM results are processed by the host and the unaligned short reads are streamed to the accelerator after reconfiguration. This allows a higher achievable population of string matchers, as no additonal memory controllers are required for storing and accessing the unaligned short reads. Furthermore, multiple reconfigurations (for cases where the on-board memory size is insufficient to store all the unaligned short reads) are not needed. We find that for large designs, the accelerator device can be reconfigured in around 115ms. If the reconfiguration time is increased 10 times, the resulting performance for D3 would be $481 Mbps$.

In our testing, D3 has the highest *bps* value due to the increased number of string matchers in each alignment phase. Table IV indicates that D3 can be up to 293 times faster than BWA [15] on the CPU, and 134 times faster than SOAP3 on the GPU. Note that the performance result of D3 is a) an estimate, while the results of [11], [16], [17] could be measured from full implementations, and b) dependent on various assumptions, such as the reconfiguration time, the achievable population and the achievable clock frequency for a given population. Note that the memory bandwidth of the system can also reduce the performance improvement when higher populations of string matchers are achieved. Since the ESM and ASM rely heavily on random access to external memory, the memory bandwidth of a system can easily become a bottleneck. For a more conservative performance estimate, the achievable population of the ESM and ASM is halved and the clock frequency is reduced to 100MHz. The resulting performance for D3 is $129Mbps$, which is still a respectable figure for a single FPGA implementation.

In addition to runtime, another important attribute of an aligner is sensitivity - the percentage of short reads that can be successfully mapped to the reference sequence. Figure 5 compares the sensitivity of our design to various software aligners. When running the software aligners for this test, the options which improve sensitivity at the cost of performance, such as --best for Bowtie and -R for BWA, are used when available.

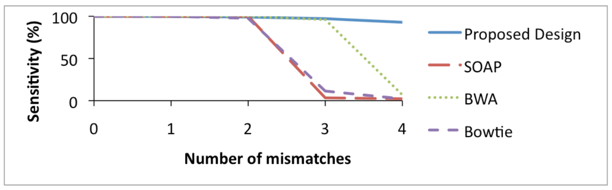

Figure 5. Sensitivity comparison for 90-base short reads.

The sensitivity values in Figure 5 indicate that for up to two mismatches, our design has comparable sensitivity to the software aligners. For short reads with more than than two mismatches, the sensitivity of the software aligners sharply decreases. This is a result of the aligner being unable to explore the large search space within the cut off time. The proposed design is able to significantly reduce the search space by only performing local alignment on the CALs most likely to map to the reference sequence (i.e. the most frequently occurring). As a result, the sensitivity is significantly better than the software aligners for greater than two mismatches. The sensitivity of our design can be improved by decreasing the seed length. However, this increases the hardware resources required for the CAL table as more CALs are found per seed, potentially limiting the achievable level of parallelism by reducing the number of ASMs able to populate the FPGA.

VI. Conclusion

In this work, we demonstrate that a carefully designed architecture on an FPGA platform can achieve promising high throughput for short read alignment. By optimising the processor responsible for the most common workload and trading off between storage and computation resources, the efficiency of our designs is significantly enhanced. Current and future research includes further optimisation of our designs and their application in clinical procedures.

Acknowledgement

We thank Dennis Lo and Rossa Chiu for their advice and encouragement. This work was supported in part by Maxeler University Programme, Xilinx, UK EPSRC, the European Union Seventh Framework Programme under grant agreement number 257906, 287804 and 318521, and the HiPEAC NoE.

References

[1] P. Ferragina and G. Manzini, "An experimental study of an opportunistic index," in *Proc. SODA*, 2001, pp. 269–278.

[2] T. B. Preuer, O. Knodel, and R. G. Spallek, "Short-read mapping by a systolic custom FPGA computation," in *Proc. FCCM*, 2012, pp. 169–176.

[3] B. Langmead *et al.*, "Ultrafast and memory-efficient alignment of short DNA sequences to the human genome," *Genome Biology*, vol. 10, no. 3, pp. R25+, 2009.

[4] R. Li *et al.*, "SOAP2: an improved ultrafast tool for short read alignment," *Bioinformatics*, vol. 25, no. 15, pp. 1966–1967, August 2009.

[5] M. Burrows and D. Wheeler, "A block-sorting lossless data compression algorithm," Digital Equipment Corporation, Tech. Rep., 1994.

[6] T. Lam, R. Li, A. Tam, S. Wong, E. Wu, and S. Yiu, "High throughput short read alignment via bi-directional BWT," in *Proc. BIBM*, nov. 2009, pp. 31 –36.

[7] S. F. Altschul *et al.*, "Basic local alignment search tool," *Molecular Biology*, vol. 215, no. 3, pp. 403–410, 1990.

[8] T. F. Smith and M. S. Waterman, "Identification of common molecular subsequences," *Molecular Biology*, vol. 147, no. 1, pp. 195–197, March 1970.

[9] E. Fernandez, W. Najjar, and S. Lonardi, "String matching in hardware using the FM-index," in *Proc. FCCM*, 2011, pp. 218–225.

[10] W. Tang *et al.*, "Accelerating millions of short reads mapping on a heterogeneous architecture with FPGA accelerator," in *Proc. FCCM*, 2012, pp. 184–187.

[11] C. B. Olson *et al.*, "Hardware acceleration of short read mapping," in *Proc. FCCM*, 2012, pp. 161–168.

[12] J. Arram, K. H. Tsoi, W. Luk, and P. Jiang, "Hardware acceleration of genetic sequence alignment," in *ARC*, 2013, pp. 13–24.

[13] [Online]. Available: http://hgdownload.cse.ucsc.edu/goldenpath/hg18/chromosomes/

[14] C. Liu *et al.*, "SOAP3: Ultra-fast GPU-based parallel alignment tool for short reads," *Bioinformatics*, vol. 28, no. 6, pp. 878–879, 2012.

[15] H. Li and R. Durbin, "Fast and accurate short read alignment with burrows wheeler transform," *Bioinformatics*, vol. 25, no. 14, pp. 1754–1760, 2009.

[16] E. Fernandez, W. Najjar, S. Lonardi, and J. Villarreal, "Multithreaded FPGA acceleration of DNA sequence mapping," in *Proc. HPEC*, 2012.

[17] Y. Chen, B. Schmidt, and D. Maksell, "An FPGA aligner for short read mapping," in *Proc. FPL*, 2012, pp. 511–514.