Hardware Acceleration of Genetic Sequence Alignment

J. Arram[1], K.H. Tsoi[1], Wayne Luk[1], and P. Jiang[2]

[1] Department of Computing, Imperial College London, United Kingdom
[2] Department of Chemical Pathology, The Chinese University of Hong Kong, China

Abstract. Next generation DNA sequencing machines have been improving at an exceptional rate; the subsequent analysis of the generated sequenced data has become a bottleneck in current systems. This paper explores the use of reconfigurable hardware to accelerate the short read mapping problem, where the positions of millions of short DNA sequences are located relative to a known reference sequence. The proposed design comprises of an alignment processor based on a backtracking variation of the FM-index algorithm. The design represents a full solution to the short read mapping problem, capable of efficient exact and approximate alignment. We use reconfigurable hardware to accelerate the design and find that an implementation targeting the MaxWorkstation performs considerably faster and more energy efficient than current CPU and GPU based software aligners.

1 Introduction

DNA contains a long sequence of pairs of nucleotide bases which can be abstracted into a character string with an alphabet {'A', 'C', 'G', '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 the genetic information of a patient 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 breaking the long DNA chain randomly. As a consequence of this action, the position and orientation information of the fragments with respect to the sample is lost. Based on the assumption that all DNA sequences within a species are similar, the sample DNA can be reconstructed by determining the location of the short fragments (the 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 recently been improving at a rate faster than Moore's law. 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 grow as fast as the speed of sequencing machines. 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. However, irregular computation patterns can affect the efficiency of FPGA accelerators. Most hardware designs overcome this problem by running large portions of the alignment algorithm on a CPU. For example, in hardware designs based on Smith-Waterman local alignment, the traceback step is either run on a CPU or ignored. As a result the performance and functionality of these aligner designs are significantly reduced.

In this paper, we propose a hardware accelerated alignment processor based on a backtracking FM-index algorithm. The design represents a full solution to short read mapping, capable of both exact and approximate alignment. The design can be fully mapped into hardware, maximising the hardware efficiency, while reducing the runtime and costs. The major contributions of our work include:

– A hardware design for a novel sequence alignment processor based on a backtracking FM-index algorithm. Various optimisations, such as those for memory size, memory bandwidth and latency and are discussed (Section 3).
– An implementation of the proposed design, showing how the design can be realised on the Maxeler MAX3 board [1] (Section 4).
– Performance evaluation of the proposed design, with comparisons against some of the fastest software solutions on multi-core processors and GPUs, as well as hardware solutions on FPGAs (Section 5).

2 Background and Related Work

FM-Index. Our design is based on FM-index [2], 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 (SA) with the Burrows-Wheeler transform (BWT) [5] , to provide an efficient method for finding all occurrences of a pattern within a long reference sequence. First the BWT of the reference sequence is generated using the following four steps:

1. Terminate the reference sequence R with a unique character: \$, which has the smallest lexicographical value.
2. Generate all rotations of R.
3. Sort the rotations lexicographically.
4. The BWT sequence is the last column of all the entries in the sorted list.

Table 1(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 SA, which indicates the the starting position of each possible suffix in R.

After generating the BWT sequence and the SA representation of the reference sequence, the functions $c(x)$ and $s(x, i)$ are defined. $c(x)$ (frequency) is the number of symbols in the BWT sequence that are lexicographically smaller than x and $s(x, i)$ (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 1(b) illustrates the $c(x)$ and $s(x, i)$ functions for the reference sequence R = ACTAGCTA.

Table 1. **(a)** BWT generation and SA representation of reference sequence R. **(b)** Values of functions $c(x)$ and $s(x, i)$ functions for the reference sequence R.

(a)

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

(b)

$BWT(R)$ = ATT\$GAACC					
$s(x, i)$	x				
i	\$	A	C	G	T
0	0	1	0	0	0
1	0	1	0	0	1
2	0	1	0	0	2
3	1	1	0	0	2
4	1	1	0	1	2
5	1	2	0	1	2
6	1	3	0	1	2
7	1	3	1	1	2
8	1	3	2	1	2
$c(x)$	0	1	4	6	7

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 a 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 (a backwards search).

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

$$l_{new} = c(x) + s(x, l_{current}) - 1 \quad (2)$$

Figure 1 shows an example of searching the pattern Q = GCT in the reference sequence R = ACTAGCTA. First, k and l are initialised to 0 and 8 respectively. Then equations 1 and 2 are applied three times, corresponding to the number of characters in Q. After the third iteration, k and l both become 6. Since $k \leq l$, the pattern can be found in the reference sequence. Note that if $k > l$ for an iteration, the symbol cannot be aligned to the reference sequence (a mismatch). The suffix array elements corresponding to each index within the SA interval give the location of the pattern in the reference sequence. Table 1(a) indicates that index 6 maps to position 4 in the reference sequence. Notice

R = ACTAGCTA Q = GCT

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

$\therefore$ SA interval = [6, 6]

Fig. 1. Example of searching the pattern Q = GCT in the reference sequence R

that the time complexity for finding *all* the matching locations is linear in length of the pattern and independent of the length of the reference sequence.

Related Work. There are several hardware accelerators for genetic sequence analysis. In [6], an FM-index based algorithm is proposed for FPGAs. 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 [7], 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 (i.e. 2.36K reads per second). In [8], 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 (i.e. 145.4K reads per second) using a Xilinx XC5V-LX330 FPGA. In [9], 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. Our work differs from previous designs since it is the first implementation of a hardware accelerated short read aligner based on a backtracking version of the FM-index. The proposed design represents a full solution to the short read mapping problem, capable of efficient exact and approximate alignment.

3 Alignment Processor Design

We propose an alignment processor design with the following features:

- A backtracking version of the FM-index algorithm with a data structure that supports both forward and backward search, which is capable of exact and approximate alignment (Section 3.1).
- A novel scheme to reduce the memory size of the FM-index occurrence array, allowing it to be stored directly on the accelerator board (Section 3.2).
- A new method to reduce external memory access frequency, while maximising memory bandwidth utilisation (Section 3.3).
- A novel scheme to process batches of short reads in parallel, maximising the throughput of the design (Section 3.4).

3.1 Backtracking Design

As discussed in Section 2, FM-index provides a method for finding all the occurrences of a pattern within a reference sequence. By incorporating backtracking into FM-index, the method is extended to allow for permutations of the pattern to be matched to the reference sequence. This allows the detection of Single Nucleotide Polymorphisms (SNPs), which represent a mismatch in a single nucleotide between the short read and reference sequence. The alignment processor supports the approximate alignment of short reads to a reference sequence within a permitted number of mismatches. This is a depth first search, in which the algorithm attempts to align each symbol in the short read to the reference sequence. When a symbol in the short read cannot be aligned (a mismatch) a different symbol is attempted. Only when the number of mismatches exceeds the permitted number does the algorithm backtrack to the previous symbol and attempts a different one. In the proposed design, only the first alignment solution (if any) is reported. The algorithm can be revised to report more solutions at the expense of performance.

The backtracking FM-index algorithm is extensively used in software aligners; however its use in hardware aligners is largely unexplored due to the complex mapping process. Backtracking algorithms are typically expressed as recursive functions in software. In hardware the circuit represents the entire computation, therefore recursive features such as branching statements are difficult to implement. We overcome this difficulty by using an iterative approach, where a stack is used to store the nodes in the current search path. For the FM-index, the items that require storage in a stack are:

- The SA interval indices k and l.
- The current number of mismatches.
- The next symbol to attempt if matching fails.

For a short read of length $|Q|$ and m permitted mismatches, the storage requirement for the stack is:

$$(2 \times |Q| \times 32bits) + (|Q| \times log_2(m)bits) + (|Q| \times 2bits)$$

For example, the storage requirement is 850 bytes when $|Q| = 100$ and $m = 4$. The pseudo code in Figure 2 describes the stack operations for one iteration of the backtracking FM-index algorithm.

As discussed in Section 2, for exact pattern matching the alignment processor is run for the same number of iterations as symbols in the short read. For approximate matching, the number of iterations to run the processor for is non-trivial and depends on the short read length, the number of mismatches and their position within the short read. The position of the mismatches has the greatest impact on the number of iterations. For example, if the mismatches are at the end of the short read, a large amount of the search space must be explored, requiring a large number of iterations. The challenge here is to reduce the number of iterations required for approximate matching, thus improving the design performance. We address this challenge by adapting the bi-directional BWT [10] for our purpose. This data structure allows searching in both directions (forward and backward search) and allows switching in search direction during the alignment of a pattern. As a result the impact of the mismatch position is minimised since the search can proceed in both directions. In this approach the BWT of the reversed reference sequence and the corresponding occurrence array, $s[x][i]$, are stored, doubling the external memory requirement. In the proposed design, the short reads are processed in stages. First, all the short reads are tested for exact alignment, then the unaligned reads are tested for one mismatch using a larger number of iterations (the average number

```
// backtrack condition
if number of mismatches > permitted mismatches
or all symbols in current node attempted
      pop stack;

// get symbol for alignment
if previous symbol successfully aligned
      get symbol from short_read;
else
      get symbol from stack and update top element to next symbol;

get k and l from top of stack;

// FM-index computation (update k and l)
apply equations (1) and (2)

if symbol successfully aligned and symbol from stack
    increment number of mismatches;

if symbol successfully aligned
      push k, l and number of mismatches on stack;
```

Fig. 2. Pseudo code describing one iteration of the backtracking FM-index algorithm

of iterations required for the number of mismatches and short read length). This reprocessing of the unaligned reads is continued until the maximum permitted number of mismatches is reached. Since approximately 70-80% of short reads in a typical data set can be exactly matched to the reference sequence, and over 90% is covered by allowing one mismatch, this method stops the processing of short read data from becoming the bottleneck.

3.2 Memory Size Optimisation

The occurrence array, $s[x][i]$, stores the number of occurrences of the symbol x in the BWT sequence from the 0^{th} position to the i^{th} position. Since human DNA has an alphabet of four symbols in approximately equal proportions, a 32 bit format is required to represent the occurrences. The total size required to store the occurrence array for a 3.2G base genome is $4 \times 3.2G \times 32bits = 51.2$G bytes. This is too large for memories in most FPGA platforms. The challenge here is to reduce the memory footprint of the occurrence array such that a) it can fit on most FPGA board memories and b) multiple independent copies of this array can be associated to multiple alignment processors.

We address this challenge by storing a full range value as a marker for every d elements in the occurrence array. To reconstruct the occurrence value of $s[x][i]$, the following components are summed: 1) the lower marker value relative to position i and 2) 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

$$4 \times \frac{3.2G \times 32bit}{d} + 3.2G \times 2\ bits$$

For example, the memory footprint is reduced to 1.6G bytes when d = 64.

This technique is used in software aligners such as SOAP2 [4], but its use in hardware designs is largely unexplored. The cost of using this technique is having to count the occurrence value directly from the BWT sequence. Counting the occurrences sequentially increases the latency of the design by $d \times l$, where l is the latency of a single add operation. For large values of d this additional latency reduces the design performance. We can make use of the inherent parallelism of FPGAs to count the occurrences in parallel using a binary adder tree. As a result the additional latency of the design is reduced to $\log_2 d \times l$.

Having a larger d value will further reduce the memory footprint. However, the rate of reduction rapidly decreases as the marker array becomes smaller than the BWT sequence. On the other hand, the hardware resources and the latency of counting process increase when the value of d becomes larger. The architecture allows users to trade-off available external memory storage and available on-chip computation resources.

3.3 Memory Bandwidth Optimisation

By the nature of the FM-index algorithm, the indexing pattern to the occurrence array, $s[x][i]$, is random. So it is difficult to further reduce external memory access frequency by caching. Applying the memory size optimisation, each indexing iteration requires two 32-bit reads for the marker values and two 128-bit reads for the d-base sequence segment. Many FPGA based accelerators place a limit on the number of connections from the FPGA to external memory. The number of alignment processors that can populate the FPGA is therefore limited by the number of connections to external memory required by an alignment processor. Furthermore, many FPGA based accelerators, such as the Maxeler MAX3 board, support burst memory access such that hundreds or thousands of bits can be accessed from memory in a single read operation. Initial processor designs extract only the desired bits, discarding the remaining bits in the burst. For large burst sizes, this method of memory access results in poor utilisation of memory bandwidth. The challenges here are to: a) reduce the number of connections to external memory in order to allow a larger population of processors per FPGA and b) appropriately layout the data in external memory in order to make efficient use of the available memory bandwidth.

We address this challenge 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 memory access frequency and external memory connections by 50%. Furthermore, by adjusting the value of d, the size of interleaved segments can be optimised for burst access. For example, by adjusting d so that one interleaved segment is exactly the size of a burst, the memory bandwidth is fully utilised in each read operation.

3.4 Latency Optimisation

For a human DNA sequence, the occurrence array, $s[x][i]$, is so large that it must be stored in memory outside the FPGA device. The access to external memory usually increases the latency by many cycles. By the nature of FM-index, the computation for each iteration of the algorithm is dependent on the results from the previous iteration. This iteration interdependence, coupled with the design latency, creates a sub-optimal pipeline. The challenge here is to minimise the effect of the latency to ensure design performance is not limited by external memory access.

We address this challenge by interleaving the processing of multiple short reads. In this approach, the alignment processor contains a buffer storing a batch of short reads. For a design latency l, the buffer is able to store l short reads. In each clock cycle a symbol from a different short read is selected and propagates through the pipeline. After l cycles the result is available and the next symbol in the short read is processed. As a result, the design is fully pipelined with a throughput of one aligned base per cycle. This design achieves the equivalent of a multi-threaded program aligning multiple short reads in parallel, but with zero thread switching overhead.

This latency optimisation comes at the cost of BRAM resources required to store the additional short reads and stacks. By using only 2 bits to represent the four valid symbols in the DNA sequence rather than the standard 8-bit ASCII characters, the additional BRAM resources required for the short reads can be reduced. Furthermore, the traffic between the host and the FPGA accelerator is significantly reduced.

4 Design Architecture

Our design architecture consists of an FPGA populated with alignment processors, connected to the host processor through a software driver. Before alignment starts, the software driver transfers the BWT sequence, $s[x][i]$ and $c[x]$, to the accelerator board, where they are stored using on-chip BRAM or external DRAM. Short reads are then streamed to the alignment processors in batches given by the design latency. Each batch of short reads is processed for a number of iterations determined by the permitted number of mismatches and the short read length. The alignment results for each short read, including the SA interval, the cost and a string representation of the alignment are reported to the software driver. This architecture is illustrated in Figure 3.

Software Driver. The alignment processor design can be fully mapped to hardware, therefore the software driver has a minimal role in the design architecture. The reference sequence changes infrequently, therefore we assume that the BWT sequence, $s[x][i]$ and $c[x]$ are generated in advance. After transferring the data structures to the accelerator board and configuring the number of mismatches permitted, the short reads are streamed to the alignment processors populating the FPGA. The software driver then pauses until all the accelerator output is received. If a short read can be matched to the reference sequence the SA interval is mapped to positions in the reference sequence using a simple lookup table. This step can be performed in hardware, however we choose to perform it using a CPU in order to reduce the number of memory controllers required by each alignment processor. If a short read is unaligned it is streamed again to the alignment processors for testing with a higher number of permitted mismatches.

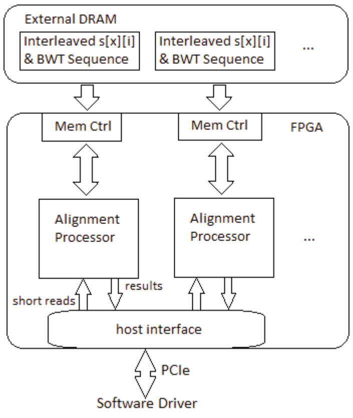

Fig. 3. Design architecture

Hardware Circuit. The FPGA is populated with alignment processors based on the backtracking FM-index algorithm described in Section 3.1. The challenge of designing the hardware circuit is mapping the iterative backtracking algorithm into hardware.

We address this challenge by implementing the stack using an array structure. In this approach, an array is used to hold the contents of the stack and the top of the stack is tracked using a pointer. The four stack items are stored in separate arrays using on chip BRAM. To implement the latency optimisation presented in Section 3.4, each array must store the stacks for l short reads, where l is the design latency. In our implementation the design latency is 58 cycles, therefore the storage requirement for the stacks is 50K bytes per processor. The pointer tracking the top element of the stacks (top), is implemented as a circular buffer using shift registers, so that its value propagates into the subsequent iteration. To push an item on the stack, top is incremented and to pop the stack top is decremented.

Using the memory size optimisation presented in Section 3.2, the occurrence array is stored in external memory with the marker values for every 64 bases. A binary tree adder is implemented to minimise the additional design latency from counting the occurrence directly from the BWT sequence. Without this optimisation the performance of the design would be reduced, since the BRAM resources required to store the additional short reads and stacks would limit the number of alignment processors able to populate the FPGA.

We implement the FM-index circuit developed in [6] for our design. The SA interval, cost and string representation of the alignment are output to the software driver when one of the following conditions is true: 1) the processor has run for the number of iterations determined by the number of mismatches permitted, 2) the entire search space has been explored and no alignment is found, or 3) an alignment solution is found. After each short read in the batch has been processed for the appropriate number of iterations, a new batch of short reads is streamed from the software driver. When all short reads have been processed, the hardware execution halts, returning control to the software driver.

5 Evaluation

In this work, we use the Human Genome version 18 [11] and short reads 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. We implement the proposed design on the Maxeler MaxWorkstation containing a MAX3 dataflow engine (DFE). The MAX3 contains a Xilinx Virtex-6 SX475T FPGA and 24GB of fast 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. Table 2 shows the resource usage for the proposed design with three alignment processors implemented in the MaxJ language. The implementation supports up to 100 bases for a single short read.

Table 2. Resource usage

Design	Registers	LUTs	BRAM	DSP
Proposed Design	20%	25%	53%	0%

The resource usage values indicate that BRAM is the critical resource in the proposed design. The maximum number of alignment processors able to populate the FPGA is seven, since a) each alignment processor uses two memory controllers and the platform supports a maximum of 15 memory controllers per FPGA and b) the resource usage values for a single alignment processor, including the resources for place and route, indicate that only 7 alignment processors can fit on the FPGA, due to the large number of BRAM resources used.

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

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

In Table 3 we compare the performance of our design implemented on the MAX3 board with existing software and hardware designs. In all our testing we process short reads with 76 bases and allow up to 2 mismatches between the short read and reference sequence.

The $baps$ values in Table 3 indicate that the proposed design is faster than current software aligners. It is over 7 times faster than BWA on an Intel X5650 CPU and 3.5 times faster than SOAP3 on an NVDIA GTX 580 GPU. To make a fair comparison to the design in [9], we estimate the performance of the proposed design implemented on theMaxeler MPC-X Series rackmount system (8 FPGA devices). The $baps$ values in Table 3 indicate that the proposed design has a comparable performance to the design in [9], even with fewer cores and a lower clock frequency. With further work into the placement of the alignment processors on the FPGA, we could reach the upper bound population and exceed the performance of the design in [9].

Table 3. Aligner performance comparison

Design	Platform	Clock freq (MHz)	Devices	Cores	*baps* (millions)
Bowtie [3]	Intel X5650	2670	1	20	1.04
BWA [12]	Intel X5650	2670	1	20	1.76
SOAP2 [4]	Intel Xeon X5650	2670	1	20	1.59
SOAP3 [13]	NVIDIA GTX 580	900	1	512	3.84
Proposed Design on MaxWorkstation	Xilinx Virtex-6 SX475T	150	1	3	13.5
Design in [9]	Xilinx Virtex-6 LX240T	250	8	8	112
Proposed Design on MPC-X (estimate)	Xilinx Virtex-6 SX475T	150	8	3	108

The energy consumption of our design is estimated using the XPower utility offered by Xilinx. It allows power analysis and estimations based on FPGA resource usage and configuration. In Table 4 the energy consumption of the proposed design is compared with existing software and hardware designs when mapping 50 million short reads.

Table 4. Energy consumption comparison

Design/Platform	Energy (W-hr)
Bowtie	19
BWA	11
SOAP2	13
SOAP3	6.3
Design in [9]	5.0
Proposed Design on MaxWorkstation (3 cores)	0.078

The values in Table 4 indicate that the proposed design consumes significantly less energy than current software aligners. This is a result of the system only drawing a small amount of power (7W) coupled with a much shorter runtime.

In addition to runtime and energy efficiency, another important attribute of an aligner is sensitivity - the percentage of short reads able to be successfully mapped to the reference sequence.

For up to two mismatches our design has a comparable sensitivity ($\sim$100%) to current software aligners. For short reads with more than two mismatches, the sensitivity of the software aligners sharply decreases ($<$20%). This is a result of the aligner being unable to explore the large search space within the cut off time.

Since the number of iterations for which the alignment processors are run for is flexible and the performance of the proposed design is better than the software aligners, we are able to process short reads for a larger number of iterations and report better sensitivities than the software aligners. To estimate the appropriate number of iterations to run the approximate matching for, we measure the average number of iterations required to align short reads of a specific length with a specific number of randomly distributed mismatches. Adjustments can be made according to the desired sensitivity.

6 Conclusion

In this work we demonstrate that an alignment processor based on the backtracking FM-Index algorithm can achieve high throughput for short read alignment applications. The design is able to map short reads to a full genome faster than current software aligners, without compromising the alignment sensitivity. Furthermore, it consumes significantly less power than software aligners, making it a feasible alternative to computational clusters. Current and future research includes further optimisation of our approach, and its application in clinical procedures.

Acknowledgements. We thank Dennis Lo and Rossa Chiu for their timely assistance and encouragement in this area of research. We also thank Maxeler University Programme, particularly Oliver Pell, and Xilinx for their support. Thanks also to ESPRC for funding our work.

References

1. `http://www.maxeler.com/products/`
2. Ferragina, P., Manzini, G.: An experimental study of an opportunistic index. In: ACM-SIAM Symposium on Discrete Algorithms (SODA 2001), pp. 269–278 (2001)
3. Langmead, B., et al.: Ultrafast and memory-efficient alignment of short DNA sequences to the human genome. Genome Biology 10(3), R25+ (2009)
4. Li, R., et al.: SOAP2: an improved ultrafast tool for short read alignment. Bioinformatics 25(15), 1966–1967 (2009)
5. Burrows, M., Wheeler, D.: A block-sorting lossless data compression algorithm. Digital Equipment Corporation, Tech. Rep. (1994)
6. Fernandez, E., Najjar, W., Lonardi, S.: String matching in hardware using the FM-index. In: Proc. FCCM, pp. 218–225 (2011)
7. Preuer, T.B., Knodel, O., Spallek, R.G.: Short-read mapping by a systolic custom FPGA computation. In: Proc. FCCM, pp. 169–176 (2012)
8. Tang, W., et al.: Accelerating millions of short reads mapping on a heterogeneous architecture with FPGA accelerator. In: Proc. FCCM, pp. 184–187 (2012)
9. Olson, C.B., et al.: Hardware acceleration of short read mapping. In: Proc. FCCM, pp. 161–168 (2012)
10. Lam, T., Li, R., Tam, A., Wong, S., Wu, E., Yiu, S.: High throughput short read alignment via bi-directional bwt. In: IEEE International Conference on Bioinformatics and Biomedicine (BIBM), pp. 31–36 (November 2009)
11. `http://hgdownload.cse.ucsc.edu/goldenpath/hg18/chromosomes/`
12. Li, H., Durbin, R.: Fast and accurate short read alignment with burrows wheeler transform. Bioinformatics 25(14), 1754–1760 (2009)
13. Liu, C.-M., et al.: SOAP3: Ultra-fast GPU-based parallel alignment tool for short reads. Bioinformatics 28(6), 878–879 (2012)