35th Annual International Conference of the IEEE EMBS
Osaka, Japan, 3 - 7 July, 2013

Implementation of a Custom Hardware-Accelerator for Short-read Mapping Using Burrows-Wheeler Alignment

Hasitha Muthumala Waidyasooriya, Masanori Hariyama and Michitaka Kameyama

***Abstract*— The mapping of millions of short DNA fragments to a large genome is a great challenge in modern computational biology. Usually, it takes many hours or days to map a large genome using software. However, the recent progress of programmable hardware such as field programmable gate arrays (FPGAs) provides a cost effective solution to this challenge. FPGAs contain millions of programmable logic gates to design massively parallel accelerators. This paper proposes a hardware architecture to accelerate the short-read mapping using Burrows-Wheeler alignment. The speed-up of the proposed architecture is estimated to be at least 10 times compared to its equivalent software application.**

Keywords: FPGA, short-read mapping, sequence alignment

I. Introduction

Next-generation sequencing machines provide millions of short DNA fragments called short-reads. Mapping such a large volume of short-reads is a fundamental aspect in modern computational biology. Complete mapping of an organism gives the possibility to understand genetic diseases, identifies cancer genomes and explores the possibilities of new drugs and personalized medicine.

Short-read mapping has been already performed using software tools such as Maq [1], Bowtie [2], BWA [3], etc using a cluster of general purpose processors (CPUs). As shown in Fig.1, short-reads with few tens to hundreds of bases are mapped to a reference genome sequence considering SNPs, insertions and deletions (indels). However, mapping of short-reads to a genome as large as a human is still a great challenge to software applications.

Recent progress of programmable hardware such as field programmable gate arrays (FPGAs) provide a cost effective solution to this challenge. FPGAs contain millions of programmable logic gates and are connected to large memories such as 4GB DDR2 and DDR3. In this paper, we propose an FPGA-based architecture to accelerate the short-read mapping using "Burrows-Wheeler alignment (BWA)" [3]. We choose BWA since it is a one of the fastest mapping tool among software methods and it provides a massive parallelism. Therefore, BWA is one of the ideal algorithm to be implemented in FPGA. The proposed architecture provides about 10 times of speed-up compared to its software implementation.

II. Related works

There are many short-read mapping software tools such as Maq [1], Bowtie [2], BWA [3] BFAST [4], SOAP2 [5], etc.

Authors are with the Graduate School of Information Sciences, Tohoku University, Aoba 6-6-05, Aramaki, Aoba, Sendai, Miyagi, 980-8579, Japan (e-mails:{hasitha, hariyama, kameyama}@ecei.tohoku.ac.jp

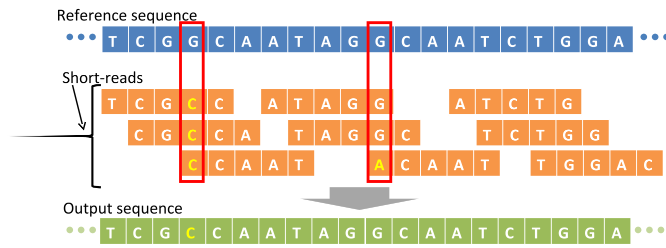

Fig. 1: Short-read mapping

Maq and BFAST use the fact that individual genomes differ only slightly and some shorter subsequences called "seeds" of a short-read will exactly match the seeds of the reference genome. In these methods, an index of the reference genome is compiled, which maps every seeds to the location where they occur. To align a short-read, all the seeds in the short-read are looked up in the index. These methods are usually more accurate but take a large processing time. Bowtie, SOAP2 and BWA use "Burrows-Wheeler Transform (BWT)" [6]. These methods are usually fast but have a less accuracy.

FPGA-based short-read mapping is proposed in [7] and [8]. The work in [7] uses a very simple method that directly compares a short-read with the reference sequence to give a position under a given number of mismatches. The reference sequence is streamed in to the FPGA and the comparison with short-reads is done in parallel. However, this method does not support indels and the processing speed is similar to some of the fast software approaches. The work in [8] implements the algorithm used in BFAST software [4] on FPGA. BFAST is more accurate than BWA but takes a large processing time.

Among the short-read mapping methods, BWA is one of the fastest in software and also gives a very good accuracy comparable to Maq. It also supports indels and contains very simple calculations such as addition, subtraction and comparisons of 32bit data. Since the calculations are very simple, it is such a waste to use CPUs or GPUs that are designed for more complex calculations such as floating-point operations, etc. Moreover, we could hardly found an article that proposes a hardware acceleration of BWA. In this paper, we propose a very simple but highly parallel architecture that uses BWA for short-read mapping.

III. Architecture of the hardware-accelerator

A. Burrows-Wheeler alignment (BWA) algorithm

In this section, we briefly describe the BWA algorithm given in [3]. It uses the "exact matching" method explained in [9]. According to [9], if a string W is a substring of the

```
Input:  BWT string B for reference string X
        Array C(.) and O(.,.) from B
        BWT string B' for the reverse of reference X
        Array O'(.,.) from B'
Output: Suffix-array intervals
Procedures:
Calculated(W)
begin
    Calculate D(i) that gives a lower bound for the
    number of mismatches in W
end
InexRecur(W, i, z, k, l)
begin
    if  z < D(i) then
        return  φ
    end
    if  i < 0 then
        return  [k, l]
    end
    I = φ
    I = I ∪ InexRecur(W, i − 1, z − 1, k, l)
    for  each b ∈ {A, C, G, T} do
        k_b = C(b) + O(b, k − 1) + 1
        l_b = C(b) + O(b, l)
        if k_b ≤ l_b then
            I = I ∪ InexRecur(W, i, z − 1, k_b, l_b)
            if b = W[i] then
                I = I ∪ InexRecur(W, i − 1, z, k_b, l_b)
            else
                I = I ∪
                InexRecur(W, i − 1, z − 1, k_b, l_b)
            end
        end
    end
    return  I
end
main(W, z)
begin
    Calculated(W)
    InexRecur(W, i, z, k, l)
end
```

Algorithm 1: Short-read mapping algorithm

string X and $k(aW) \leq l(aW)$, string aW is also a substring of X where aW equals the string $\{a, W\}$. The terms k and l, given by Eqs.(1) and (2) respectively, are the lower and upper bounds of the suffix array (SA) interval of X.

$$k(aW) = C(a) + O(a, k(aW) - 1) + 1 \tag{1}$$

$$l(aW) = C(a) + O(a, l(aW)) \tag{2}$$

The number of symbols that are lexicographically smaller than a is given by $C(a)$. The number of occurrence of a in string $B[0, i]$ is given by $O(a, i)$. The BWT array of the string X is given by B and the part of the string B that includes i number of symbols from the beginning is given by $B[0, i]$.

Position	0	1	2	3	4	5	6
Ref. sequence (*X*)	C	C	T	G	A	G	$

Position	0	1	2
Short read (*W*)	C	G	A

a	A	C	G	T
C(*a*)	1	2	4	6

(a) Reference sequence X, short-read W and $C(a)$ of X

Position	0	1	2	3	4	5	6
0	C	C	T	G	A	G	$
1	C	T	G	A	G	$	C
2	T	G	A	G	$	C	C
3	G	A	G	$	C	C	T
4	A	G	$	C	C	T	G
5	G	$	C	C	T	G	A
6	$	C	C	T	G	A	G

(b) Rotation of X

BWT string (B)
Occurrence array O(. , .)

SA	Position	0	1	2	3	4	5	6	$	A	C	G	T
0	6	$	C	C	T	G	A	G	0	0	0	1	0
1	4	A	G	$	C	C	T	G	0	0	0	2	0
2	0	C	C	T	G	A	G	$	1	0	0	2	0
3	1	C	T	G	A	G	$	C	1	0	1	2	0
4	5	G	$	C	C	T	G	A	1	1	1	2	0
5	3	G	A	G	$	C	C	T	1	1	1	2	1
6	2	T	G	A	G	$	C	C	1	1	2	2	1

(c) Sorting result

Fig. 2: Mapping example

Algorithm 1 shows the short-read mapping method in [3]. To explain the algorithm, let us consider the example shown in Fig.2. We have a reference sequence X and a short-read W as shown in Fig.2(a). According to the Algorithm 1, the inputs are the BWT of X and X', occurrence arrays $O(.,.)$, $O'(.,.)$ and $C(.)$. The array $C(.)$ and the rotation of X are shown in Figs.2(a) and 2(b) respectively. The sorting result is shown in Fig.2(c). We can see the BWT and occurrence arrays B and $O(.,.)$ in Fig.2(c). Similarly, we calculate B' and $O'(.,.)$ for X', which is the reverse sequence of X. We skip the explanation of procedure "Calculated" in Algorithm 1 that gives a lower bound $D(.)$ for the number of mismatches in W. For more details, please refer [3].

Figure 3 shows the mapping of the short-read W. The procedure "InexRecur(W, i, z, k, l)" is called after calculating the lower bound $D(i)$. The search position of W, the number of mismatches allowed (SNP and indel), the lower and upper bounds of the suffix array are given by i, z, k and l respectively. Figure 3 shows all executions of the procedure "InexRecur". Note that, the term "(i, z, k, l)" in Fig.3 represents the procedure "InexRecur(W, i, z, k, l)". After the mapping is finished, we get the position of the short-read and its SNP and indel data. In this example, we get the result in Fig.4 considering both SNPs and indels as mismatches. The result with one insertion has an SA interval of $(5, 5)$ as shown in Fig.3. According to Fig.2(c), SA interval $(5, 5)$

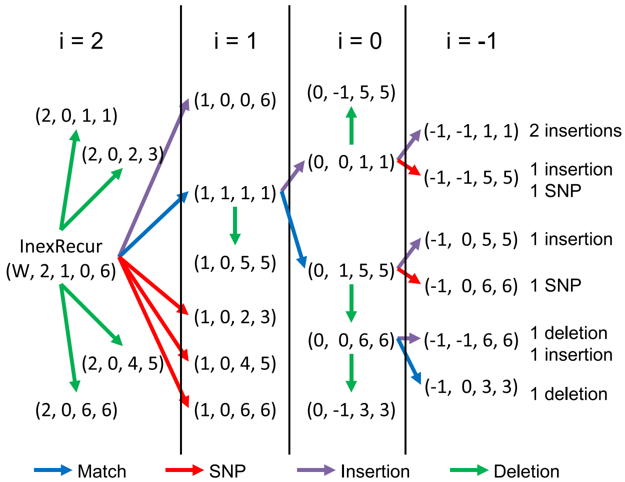

Fig. 3: Mapping of short-read W

Position	0	1	2	3	4	5	6
X	C	C	T	G	A	G	
W				C	G	A	
Result	C	C	T	C	G	A	G

(a) SA interval : (5,5), position : 3 Insertion @ X(3)

Position	0	1	2	3	4	5	6
X	C	C	T	G	A	G	
W			C	G	A		
Result	C	C	C	G	A	G	

(b) SA interval : (6,6), position : 2 SNP @ X(2)

Position	0	1	2	3	4	5	6
X	C	C	T	G	A	G	
W		C	G	A			
Result	C	C	G	A	G		

(c) SA interval : (3,3), position : 1 Deletion @ X(2)

Fig. 4: Mapping results after allowing one mismatch

refers to the position 3 in reference X. Therefore, we map W at 3 as shown in Fig.4(a). The result with one SNP has a SA interval of $(6, 6)$. Therefore, we map W at 2 as shown in Fig.4(b). Similarly, one deletion with SA interval $(3, 3)$ is mapped at 1 as shown in Fig.4(c). As shown in Fig.3, procedure "InexRecur" can be executed in "depth-first" or "breadth-first" or a hybrid of both. We use this behavior in our hardware architecture explained in the next section.

B. FPGA accelerator architecture

In practical problems, the same reference genome sequence is used to map different short-reads. Therefore, in this paper, we perform the Burrows-Wheeler transform and calculate the occurrence array offline and just transfer the data to the FPGA for different mappings. Figure 5 shows the accelerator architecture. It consists of a very simple 1-dimensional 128 processing element (PE) array and two DDR2 memories. The parallelism is achieved by mapping different short-reads on 128 PEs simultaneously. The occurrence array and short-reads are transferred to the DDR2 memory and the FPGA internal memory respectively.

Structure of a PE is given in Fig.6. It consists of a 32bit adder, a comparator and pipeline registers to perform the calculations explained in Eqs.(1) and (2). After finishing one "InexRecur" procedure, a new one is loaded from the

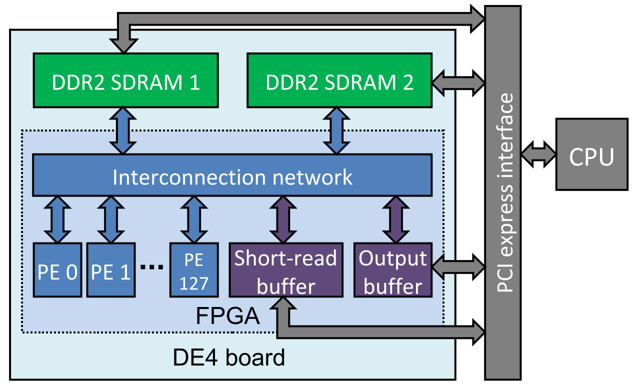

Fig. 5: Accelerator architecture

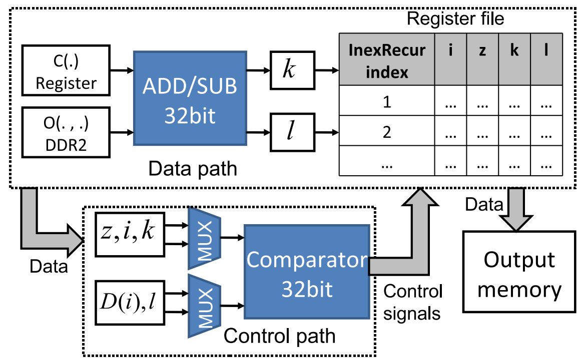

Fig. 6: Structure of a PE

register file. In each "InxRecur" procedure, new calls to the same procedure are generated as explained in Algorithm 1. The parameters of such recursive calls are stored in the register file, so that we can keep a track of all the recursive calls. The "ADD/SUB" unit in PE is used to calculate the Eqs.(1) and (2). The comparator and the control path do all the conditional branches in the "InxRecur" procedure. New short-reads are fed to the PEs after the old short-reads are mapped. The mapping result is read by the CPU. Note that, CPU reads and empties the output buffer before it overflows, so that new mapping results can be written to it. Moreover, CPU transfers short-read data gradually while there are being mapped. Unlike the CPU that has a complex floating-point ALU and very complicated control circuit, PE is a very simple unit that specialized only to map a short-read. It is designed using minimum resources. Therefore, we can have a lot of PEs in the same FPGA to provide a comparable performance to a super computer that has many CPUs.

If the accelerator has many PEs, at least one PE wants to access the DDR2 memory in each clock cycle as shown in Fig.7. Therefore, the total performance depends on the DDR2 access speed. Note that, the memory access in short-read mapping is random since it heavily depends on the short-read data. Although CPUs have a theoretical bandwidth of over 50MBps, it is drastically reduced when the memory access is random. However, in FPGAs, we can maintain a very high memory access speed even in random access by designing custom address generation units and pipelined data-paths. As a result, the memory access in FPGAs could be faster by the factor of few tens.

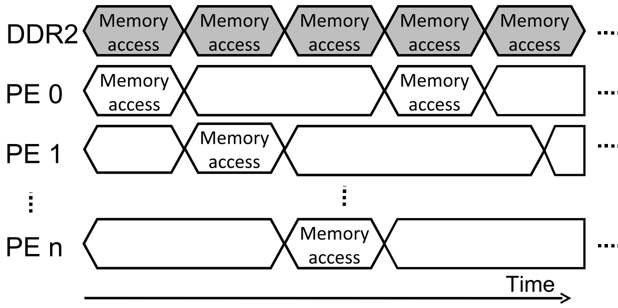

Fig. 7: Memory access in FPGA

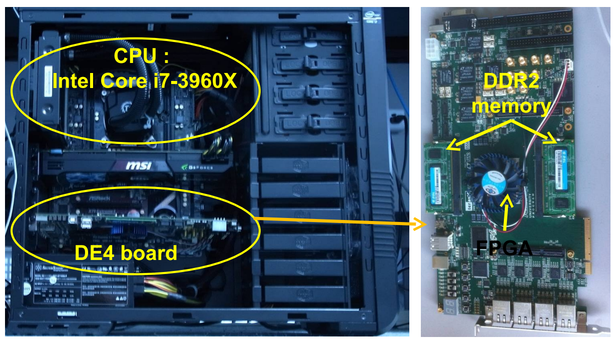

Fig. 8: Computing system

IV. Evaluation

For the evaluation, we used DE4 board [10] that contains “Altera EP4SGX530KH40C2 FPGA” and two 4GB DDR2-SDRAMs. The system shown in Fig.8 contains a core i7-3960x CPU and a DE4 board connected through the PCI express port. The operating frequency of the accelerator is 100MHz. Table I shows the resource usage of a PE. As shown in Table I, the PE size is limited by the registers and we can have more than 400 PEs on a single FPGA.

Table II shows the memory bandwidth comparison of the CPU-based system and the FPGA-based system. Although CPU has a better theoretical bandwidth, it reduces drastically for random data access. Since the memory access in short-read mapping is random, small bandwidth is a big problem in CPU or GPU based accelerations. However, in FPGA, we can design custom address generation units for random access. Therefore, even in random access, we can achieve 5.15GB/s bandwidth which is very close to the theoretical one. Since there are two DDR2 memories in FPGA, we can achieve a bandwidth of over 10GB/s by accessing them in parallel. Therefore, the memory access speed is more than 10 times in the FPGA compared to the CPU. Therefore, we can assume the proposed system is at least 10 times faster than a CPU-based system. Note that, we can use up to 3 FPGA boards in one computer to increase the processing power. We can also connect many such computers to design a super computer with FPGAs. Moreover, different FPGAs can be

TABLE I: Resource usage of a PE

Resource name	Usage	usage/total_ available_resources $\times 100\%$
LUTs	844	0.20
Registers	1015	0.24
Onchip memory	1.25kB	0.05

TABLE II: Memory access speed

	Bandwidth (GB/s)	
	Theoretical	Random access
CPU (i7-3960x)	51.2	1.06
FPGA (EP4SGX530KH40C2)	6.4×2	5.15×2

connected directly to decrease the communication overhead. Therefore, this FPGA-based system has a great potential to dramatically increase the processing speed.

Currently, we have successfully mapped 100 short-reads by using small reference string that have few thousands of symbols. The speed-up factor compared to software is measured around 10 times and the accuracy is as the same as that in BWA software. With these results, we can say that the proposed system works without any problem. In future works, we are expecting to provide experimental results using large references such as a human genome.

V. Conclusion

We have proposed a hardware accelerator architecture for BWA algorithm to increase the processing speed. The algorithm is successfully implemented on FPGA with an estimated 10 times speed-up compared to the software. The FPGA used is a mid-range less expensive one. It is possible to increase the processing power by choosing a latest FPGAs such as Altera Stratix V. Moreover, we can use multiple FPGAs in a single PC and also a cluster of such PCs to increase the processing speed massively.

Acknowledgment

This work is supported by MEXT KAKENHI Grant Number 12020735.

References

[1] H. Li, “Maq: Mapping and assembly with qualities”, http://maq.sourceforge.net/, 2008.

[2] B. Langmead, C. Trapnell, M. Pop, and S. Salzberg, “Ultrafast and memory-efficient alignment of short dna sequences to the human genome”, Genome Biology, Vol.10, No.3, p.R25, 2009.

[3] Heng Li and Richard Durbin, “Fast and accurate short read alignment with Burrows-Wheeler transform”, Bioinfomatics, Vol.25, No.14, pp.1754-1760, 2009.

[4] N. Homer, B. Merriman and S. F. Nelson, “BFAST: An Alignment Tool for Large Scale Genome Resequencing”, PLoS ONE, vol.4, No.11, p.e7767, 2009.

[5] R. Li, C. Yu, Y. Li, T. Lam, S. Yiu, K. Kristiansen and J. Wang, “Soap2: an improved ultrafast tool for short read alignment” Bioinformatics, Vol.25, No.15, p.1966, 2009.

[6] M. Burrows and D. J. Wheeler, “A block-sorting lossless data compression algorithm”, Digital Equipment Corporation, Palo Alto, CA, Technical report 124, 1994.

[7] O. Knodel, T. B. Preuser and R. G. Spallek, “Next-Generation Massively Parallel Short-Read Mapping on FPGAs”, IEEE Int’l Conf. on Application-Specific Syst., Archi. and Processors, pp.195-201, 2011.

[8] C. B. Olson, M. Kim, C. Clauson, B. Kogon, C. Ebeling, S. Hauck, and W. L. Ruzzo, “Hardware Acceleration of Short Read Mapping”, IEEE 20th Annual Int’l Symp. on Field-Programmable Custom Computing Machines, pp.161-168, 2012.

[9] P. Ferragina and G. Manzini, “Opportunistic data structures with applications”, Proc. of 41st Symp. on Foundations of Computer Science, pp.390-398, 2009.

[10] http://www.altera.com/education/univ/materials/boards/de4/unv-de4-board.html