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

An Optimized and Low-cost FPGA-based DNA Sequence Alignment – A Step towards Personal Genomics

Hurmat Ali Shah, Laiq Hasan, Nasir Ahmad

Abstract— **DNA sequence alignment is a cardinal process in computational biology but also is much expensive computationally when performing through traditional computational platforms like CPU. Of many off the shelf platforms explored for speeding up the computation process, FPGA stands as the best candidate due to its performance per dollar spent and performance per watt. These two advantages make FPGA as the most appropriate choice for realizing the aim of personal genomics. The previous implementation of DNA sequence alignment did not take into consideration the price of the device on which optimization was performed. This paper presents optimization over previous FPGA implementation that increases the overall speed-up achieved as well as the price incurred by the platform that was optimized. The optimizations are (1) The array of processing elements is made to run on change in input value and not on clock, so eliminating the need for tight clock synchronization, (2) the implementation is unrestrained by the size of the sequences to be aligned, (3) the waiting time required for the sequences to load to FPGA is reduced to the minimum possible and (4) an efficient method is devised to store the output matrix that make possible to save the diagonal elements to be used in next pass, in parallel with the computation of output matrix. Implemented on Spartan3 FPGA, this implementation achieved 20 times performance improvement in terms of CUPS over GPP implementation.**

I. INTRODUCTION

Sequence alignment is a resource-intensive operation, compelling researchers to search out for a dedicated and suitable platform as the general purpose CPU could not be optimized to peculiarities of sequence alignment. The collusion of high throughput generated by next generation sequencing (NGS) technologies, exact sequence alignment techniques and fast as well as cheap reconfigurable platforms have converted the dream of personal genomics into a possibility.

The tested and trusted Smith-Waterman algorithm is used on a reconfigurable system for aligning data generated by NGS technologies. In [1], a comparison between three Field Programmable Gate Array , Graphical Processing Unit and Cell Broadband Engine reconfigurable platforms for aligning biological sequences based on four parameters i.e. speed, energy consumption and developmental costs has been presented. They showed that FPGA outperform the other two platforms in terms of performance per watt and performance per dollar spent. The advantage of FPGA in high performance computing is that, if used as accelerator, computing density can be increased significantly and also FPGA can benefit from the increased clock speed as opposed to general microprocessors which have reached their maximum clock speed.

This paper presents an efficient FPGA model for Deoxyribonucleic Acid sequences alignment which employs different optimization techniques for performing the required task on a low-price FPGA, making headway towards personal genomics. Many of the high performance implementations of DNA sequence alignment are available but all of them face certain limitations. Besides the high cost of the implementation platform, some are restricted by sequence query length while others suffer from inefficient use of the available memory. [2] Worked on reducing the loading time of alignment matrix columns for subsequent reuse by storing the columns on each processing element (P.E.). But this lowered the space available for instantiating PEs. [3] Designed a 2D systolic array for achieving a speed-up of 160x for aligning two sequences. However there are two main restrictions to this implementation i.e. the size of the sequences to be aligned that was equal to the size of systolic array and the PEs have to spend idle time until a new sequence was loaded from DDRAM to the local memory of FPGA. [4] Also has maximum sequence length limitation as the sequences larger than 8192 characters are to be aligned on host PC. [5] Used CREC reconfigurable computer for speeding-up the biological sequence alignment but due to implementation in high level language lost sight of hardware level optimizations. [6] Provided multiple improvements over the available designs like multiple stage PE, pipelined mechanism that can support uneven stage latencies and the compressed substitution matrix storage structure. But the incorporation of these optimizations on the Cray XDI 100 supercomputing platform also disfavored its wide-scale use as it was not an inexpensive option. MGAP [7], Kestrel [9] and Fuzion are based on concept of SIMD architecture. [10] And [11] focused on efficiently designing the PE for implementing the alignment of two sequences based on smith-waterman algorithm. Some of the implementations such as [6] and also [10] focused on tailoring the PE design for various parameters of the sequences to be aligned.

Our implementation uses the highly efficient Smith-Waterman systolic cell designed by [12]. Linear systolic array is implemented instead of 2D array because globally asynchronous timing can be employed to exploit the idle time the hardware spends after the end of first run and

Hurmat Ali Shah is M.Sc student at Department of Computer Systems Engineering, UET Peshawar. 2500 Pakistan; (e-mail: hurmat_engr@yahoo.com).

Laiq Hasan is with the department of Computer Systems Engineering, University of Engineering and Technology Peshawar, Pakistan (e-mail: laiqhasan@nwfpuet.edu.pk).

Nasir Ahmad is with the University of Engineering and Technology Peshawar Pakistan working as a Assistant Professor at the department of Computer Systems Engineering (e-mail: n.ahmad@nwfpuet.edu.pk).

waiting before the second run can be started of the 2D systolic array. The linear systolic array is made to operate asynchronously i.e. dependent on data changes and not on strict clock synchronization. This saves much development time and also enables the system to run on signal propagation time of the circuit and eliminate the overestimate of computation time caused by employing clock [12]. The second optimization of our implementation is that it is not restrained by the sequence lengths and can align sequences of any length in multiple passes. The multi-pass strategy of ours need not to communicate with host PC every time the last of PE in the systolic array calculates the last alignment score but instead fetches new characters from local SRAM and the PEs start again processing. So our system employs two different types of passes; one is from host PC to SRAM of FPGA and another from the SRAM to PEs. The third improvement of our implementation is that our system need not to sit idle until all characters of the sequences are loaded to SRAM of FPGA, instead our system start aligning the two sequences at the instant when one character from each of the sequences arrives at SRAM and are transferred to the first PE for calculating the alignment score. The fourth optimization is the novelty introduced for storing the last column of alignment matrix after one pass for subsequent use in next pass. The system is not halted for the time the last column is stored to local SRAM before the calculated score matrix is sent to host PC, but instead it is performed in parallel to the calculation of the matrix i.e. when the PEs are aligning the two sequences. This optimized design is implemented on Spartan 3 FPGA, a low-priced device, making our implementation a significant step towards realizing the aim of personal genomics.

The rest of paper is organized as; section 2 deals with system design, section 3 provides the algorithms employed for designing the system, section 4 presents results of the study while section 5 concludes the paper.

II. System Design

Smith-Waterman algorithm is a dynamic programming algorithm that finds similarity between two sequences by aligning them locally. This algorithm assigns a score to the characters assigned based on degree of similarity between the two sequences. The recursive equations are given as;

$$F(i,j) = max \begin{cases} 0, \\ F(i-1,j-1) + s(xi,yj), \\ F(i-1,j) - d, \\ F(i,j-1) - d, \end{cases}$$

$S(x_i, y_i)$ is similarity score whereas d is gap penalty while i and j are lengths of the two sequences. SW alignment algorithm has $O(MN)$ space and time complexity of computing the output matrix. To exploit the space and time saving provided by SW algorithm, [11] designed an efficient cell that computes the above equation. The cell design is given in Figure 1;

The cell design has been optimized for linear systolic array (LSA) implementation and exploits the computational edges offered by it. The design does not have extra logic for calculating the diagonal element but instead translates into design logic the fact that actually the diagonal element required for performing pair-wise alignment, is output of previous processing element delayed by one clock cycle.

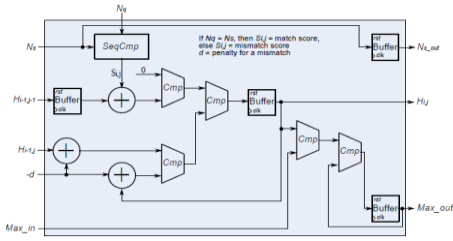

Figure 1 Cell design for SW alignment

Another spatial saving of LSA is that the up element is the current value of the PE and need not to be calculated separately. The design has used this fact and has also hardcoded '0' that saves hardware resources. For functioning of the cell see [8]. Our system implements LSA instead of 2D systolic array because of reasons given above. LSA is a linear arrangement of processing element where one PE takes input from its predecessor and passes it to its successor synchronously. In our case the cell given above was used as one PE. One PE computes one whole column of output matrix in clock cycles equal to the number of anti-diagonals in the matrix [9].

Figure 2 shows block diagram of our implementation, where each PE implements the cell given in Figure 1;

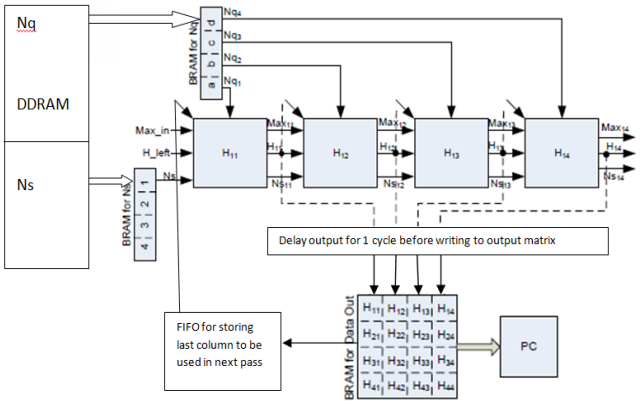

Figure 2 Block Diagram of our implementation. The corner elements for PE come from the previous pass. Output is delayed for one clock cycle to synchronize reading from and writing to the output matrix H.

Let's consider that *Ns* and *Nq* are two sequences to be aligned and there are four PEs. BRAMs allocated for the sequences have length of four. The two BRAMs receive characters from the respective portion of DDRAM that stores the whole of *Ns* and *Nq*. Four characters of each sequence from DDRAM are transferred to respective BRAM and aligned against each other. Each PE is fed with a character from *Nq* and the character stays there as long as all the characters from sequence *Ns* are not aligned with it. After it is done the characters of both BRAMs are replaced with new characters from DDRAM. The different lengths of BRAM used for sequence *Ns* had depth around 500, so the *Ns* stays there till all characters from *Nq* are aligned with it. The depth of BRAM for *Nq* is not allowed to be greater than 500 due to scarcity of memory resources. *Nq* is often a combination of many sequences or it may be whole genome while an individual sequence stored in BRAM of *Ns* may not be such long [14], [15].

The output matrix *H* is written and also read for saving the last column to be used in next pass for aligning new batch of *Nq* entries against the *Ns*. To avoid reading and writing from the same location at the same time the writing to *H* matrix is delayed for one clock cycle. For this purpose the outputs are buffered. It makes the storing of last column to run in parallel to calculating entries of matrix rather than to halt the whole system for the time the last column is transferred to another location. Obviously this time saving is achieved at the expense of hardware resources but it is worth the resources consumed. It is well illustrated by an example; If *Ns* is of length 500 and the number of PEs are 25 then after one pass the entries in H matrix are 12500, so the last column have 500 entries and to save these 500 elements the whole system has to remain idle for 500 clock cycles. After these 500 characters are aligned with 25 characters stored in the array of PEs, new batch of 25 characters are transferred to the array of PEs and the Ns BRAM address is reset to zero so the first PE can get the first character from sequence *Ns* and align it to the new character from sequence *Nq*. As the PE array is responding to the changes in *Ns* value, so no tight clock synchronization is required to have the right character in the right place. The first PE takes the diagonal elements from the FIFO that contains the last column of the previous computation. Diagonal elements to the rest of PEs come from the previous PE. Figure 3 shows operation of the system.

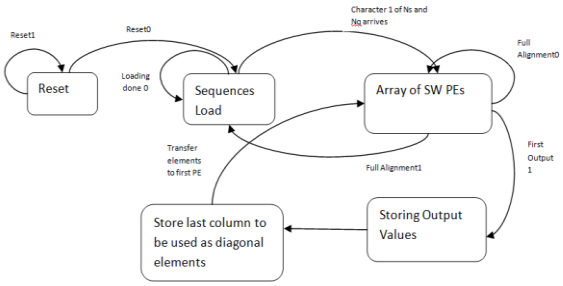

Figure 3 Execution flow diagram

Sequence Load remain active until all of the BRAMs specified for *Ns* and *Nq* are occupied, but the system does not wait for the full sequence to load to start its operation. Instead the first two characters one from both *Ns* and *Nq* are loaded to the first PE and in the cycle after that the second PE get the second character from sequence *Nq* and the first character of sequence *Ns* arrives from the first PE. In this way the m^{th} PE starts functioning in the m^{th} cycle. So the m^{th} PE is idle for m cycles after m characters from the sequence *Nq* are aligned with whole of sequence *Ns* and the $(m\text{-}1)^{th}$ PE is idle for *(m-1)* cycles. The cumulative effect can be reduced by increasing the number of PEs but our design keeps the effect to the minimum by making the sequence load and operation of the PEs array run in parallel. This is done by having two distinct addresses for reading and writing. It has complicated the designing of BRAM address control unit but has in turn blessed the system with overall time saving.

III. BRAM Address Control Unit

The hard part of the system to deal with is BRAM address control unit. To achieve the parallelism, read and write addresses need to be generated according to different conditions of the system. BRAM address control unit for sequence Ns is relatively simple as it resets the read address to zero after one pass i.e. after the last character of sequence *Ns* is aligned with last of *Nq* stored in the last PE. Calculating read and write addresses for sequence *Nq* is rather complex because the characters to be read from sequence *Nq* is dependent on the number of PEs. Figure 4 gives a block diagram representation.

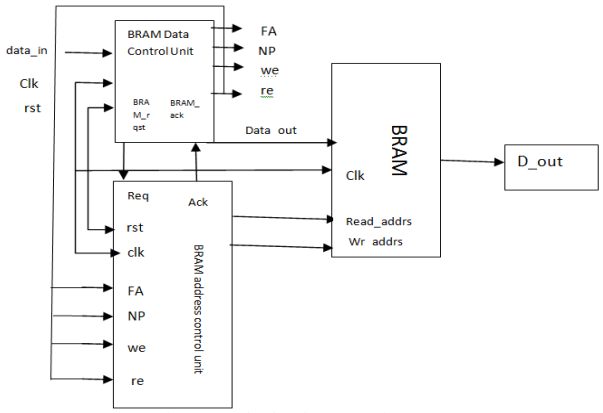

Figure 4 Block Diagram of RAM.

Following explain the flags;

- we = enabling writing the data_in to BRAM
- re = enabling reading to *D_out*
- NP = Next Pass. It is turned on when the last character of sequence Ns is aligned to the character from Nq stored in last PE.
- FA= Full Alignment. It is turned on when the last character of Ns is aligned with last character of *Nq* stored in BRAM of *Nq*.

The different algorithms used for calculating read and write addresses for both *Ns* and *Nq* are given below.

Let's consider the length of *Ns* is i and that of *Nq* is j and there are m PEs.

For reading from *Ns*
If Ns_read_addr == i and NP == On
Ns_we = Off
Ns_re = On
Ns_read_addr = 0
end if
For reading from *Nq*
if Ns_write_addr < i
Ns_write_addr = Ns_write_addr + 1
Ns_we = On
Ns_re = Off
For loading new sequence instead of Ns
If Ns_write_addr == i and FA = On
Ns_write_addr = 0;
We = On;
Re =off;
end if
For writing to Nq
if NP == Off
Nq_read_addr = Nq_read_addr + 1
Nq_we= Off
Nq_re =On
end if
For reading from Nq
if NP == On

```
    Nq_re = Off
    end if
For writing to Nq
    If Nq_write_addr < j
    Nq_write_addr = Nq_write_addr + 1
    Nq_re = Off
    Nq_we = On
    end if
Loading new sequence to Nq BRAM from DDRAM
    if FA == On
    Nq_write_addr = 0
```

IV. Results and Discussion

Our system was checked to align two array of size 500 each. Considering each character was of two bits (a DNA character can be A, C, T and G. This can be coded using two bits), a significant space of BRAM was left for storing the output matrix each element of which was 16-bit wide. This reduces the number of times the FPGA has to send the output matrix to DDRAM. The array of PEs do not have to sit idle until all of the sequence characters are loaded to FPGA, instead all PEs operate in parallel. The most often performance metric used for computational biology applications is Cells Update per Second (CUPS) and is given as;

Performance in CUPS = No. of PEs x Operating frequency

The device we used was *Spartan 3 XC3S250E*. We were able to instantiate an array of PEs of size 35 on the said device. The operating frequency shown by post place and route tool was 50 *MHz*, so the performance of our system is;

Performance = 35 x 50 MHz = 1.75 GCUPS

The performance gain is significant given the low price of the device used. The performance increases linearly with increasing the number of PEs. [1] Evaluated performance of different platforms for biological sequence alignment. Table 1 compares the CPU implementation of [1] with ours.

TABLE 1 COMPARING WITH GPP IMPLEMENTATION

Implementation	Performance
[1]	0.085 GCUPS
Ours	1.75 GCUPS
Performance Improvement	20x

[5] Was able to implement an array of PEs of size 13 on Spartan 3 1500 chip. Table 2 presents the efficiency of our design in terms of hardware utilization. Our design more efficiently utilizes hardware as it was able to realize more PEs in limited hardware resources in contrast to [5] that realized less PEs in larger hardware resources available.

TABLE 2 IMPROVEMENT IN HARDWARE UTILIZATION

Implementation	Device	Size of PE array
[5]	Spartan3 1500	13
Ours	Spartan3 XC3S250	35
Hardware utilization improvement		2.69

V. Conclusions and Future Work

This paper presented an efficient system design for aligning DNA sequences. The implementation was carried out on a low-priced FPGA that reduces the developmental cost and makes it suitable to be employed in personal genomics. Four different improvements were carried out over the available FPGA implementation that made possible alignment of large DNA sequences on limited hardware resources. The idle time of PEs was reduced to the minimum possible that resulted in faster alignment of large DNA sequences. The implementation resulted in 20 times GCUPS improvement over GPP implementation and hardware utilization was improved by 2.69 times. The design can be extended further to align a query with whole genome by increasing the number of PEs. This can be done by using an FPGA device that contains more hardware resources.

The output matrix need to be sent every time character from sequence Nq equal to the number of PEs is aligned with all characters of sequence Ns. This incurs heavy communication overhead. We aim to reduce this overhead by incorporating two improvements in our implementation. (1) The space required for storing the output matrix will be reduced by storing the elements more efficiently. (2) Improving PC ↔ FPGA communication by using faster communication techniques as Multi-Gigabit transceiver.

References

[1] Li, Heng, and Nils Homer. "A survey of sequence alignment algorithms for next-generation sequencing." Briefings in Bioinformatics 11.5 (2010): 473-483.

[2] Benkrid, Khaled, et al. "High performance biological pairwise sequence alignment: FPGA versus GPU versus cell BE versus GPP." International Journal of Reconfigurable Computing 2012 (2012): 7.

[3] Gok, Mustafa, and Caglar Yilmaz. "Efficient Cell Designs for Systolic Smith-Waterman Implementations." Field Programmable Logic and Applications, 2006. FPL'06. International Conference on. IEEE, 2006.

[4] Li, Isaac TS, Warren Shum, and Kevin Truong. "160-fold acceleration of the Smith-Waterman algorithm using a field programmable gate array (FPGA)." BMC bioinformatics 8.1 (2007): 185.

[5] Oliver, Tim, Bertil Schmidt, and Douglas Maskell. "Hyper customized processors for bio-sequence database scanning on FPGAs." Proceedings of the 2005 ACM/SIGDA 13th international symposium on Field-programmable gate arrays. ACM, 2005.

[6] Szente, Balint, et al. "A General Smith-Waterman Algorithm Implementation Using the CREC Reconfigurable Computer."

[7] Hsien-Yu Liao, Meng-Lai Yin, Yi Cheng, A Parallel Implementation of the Smith-Waterman Algorithm for Massive Sequences Searching, IEEE EMBS, 2004.

[8] Borah, M., Bajwa, R.S., Hannenhalli, S., Irwin, M.J.: A SIMD solution to the sequence comparison problem on the MGAP, in Proc. ASAP'94, IEEE CS (1994) 144-160.

[9] Di Bias, A., et al. "The UCSC Kestrel parallel processor." Parallel and Distributed Systems, IEEE Transactions on 16.1 (2005): 80-92.

[10] Oliver, Tim, Bertil Schmidt, and Douglas Maskell. "Hyper customized processors for bio-sequence database scanning on FPGAs." Proceedings of the 2005 ACM/SIGDA 13th international symposium on Field-programmable gate arrays. ACM, 2005.

[11] Yu, Chi Wai, et al. "A Smith-Waterman systolic cell." New Algorithms, Architectures and Applications for Reconfigurable Computing (2005): 291-300.

[12] L. Hasan, "Hardwar acceleration of biological sequence alignment", Ph.D. thesis, TU Delft,2011.

[13] Schmidt, B., Schröder, H., Schimmler, M: Massively Parallel Solutions for Molecular Sequence Analysis, Proc.1st IEEE Int. Workshop on High Performance Computational Biology, Ft. Lauderdale, Florida, 2002.

[14] Boeckmann, Brigitte, et al. "The SWISS-PROT protein knowledgebase and its supplement TrEMBL in 2003." Nucleic acids research 31.1 (2003): 365-370.

[15] Schuster, Stephan C. "Next-generation sequencing transforms today's biology." Nature 200.8 (2007).