IJCAES
ISSN: 2231-4946
Volume III, Special Issue, August 2013

International Journal of Computer Applications in Engineering Sciences
Special Issue on National Conference on Information and Communication (NCIC'13)
www.caesjournals.org

Implementation of Word Matching Stage of BLASTN Using Modified Bloom Filter

A.Jenifer[1], S. Karthick[2]

[1]*M.E VLSI DESIGN, SreeSastha Institute of Engineering and Technology, Chennai, India.*

[1]jenniferece11@gmail.com

***Abstract*—Basic Local Alignment Search Tool (BLAST) is a standard computer application that molecular biologists use to search for sequence similarity in genomic databases. BLASTN, a version of BLAST specifically designed for DNA sequence searches i.e., it will find the similarities between the query sequence and the subject sequence. This similarity is to understand the function and evolutionary history of an organism. In this BLAST process, word matching stage is the most time consuming part. While the database increases, the speed of the BLAST process gets decreased. In order to derive an efficient structure for BLASTN, a reconfigurable architecture is proposed to accelerate the computation of the word matching stage. This paper describes an FPGA based hardware implementation designed to accelerate the BLAST algorithm. The main objective is to explore the feasibility of using bloom filter to realize a portable FPGA based accelerator. Finally, the results are compared with the NCBI BLASTN software running on a general purpose computer.**

***Keywords*— DNA sequencing, Bloom Filter, Hashing in hardware, FPGA.**

I. INTRODUCTION

Bioinformatics is a buzzword in this modern era of scientific research. Technically, it can be said that human genes consist of block of amino acids or proteins. The main challenge of bioinformatics is to analyze these protein sequences and derive information that can be put to use for different purposes. Scanning genomic sequence database is a common and after repeated task in molecular biology. The need for speeding up these searches comes from the rapid growth of these gene banks: every year their size is scaled by a factor 1.5 to 2 [1]. The aim of a scan operation is to find similarities between the query sequence and a particular genome sequence, which might indicate similar functionality. Dynamic programming based alignment algorithm can guarantee to find all important similarities. However, as the search space is the product of the two sequences, which could be several billion bases in size. So it is not feasible for direct implementation. One approach is needed to speed up this time-consuming operation of the searching process. One of the most widely used sequence analysis tools to use heuristics is the basic local alignment search tool (BLAST)[2].BLASTIN, a version of BLAST specifically designed for DNA sequence searches, consists of a three-stage pipeline:

(i) Word Matching
(ii) Ungapped Extension
(iii) Gapped Extension

Analysis of the various stages of the BLASTN pipeline (see Table I) reveals that the word matching stage is the most time consuming pout. About 80% of the time will be spent in this stage itself. Therefore, accelerating the computation of this stage will have the greatest effect on the overall performance.

TABLE I
PERCENTAGE OF TIME SPENT IN EACH STAGE OF NCBI BLASTN

Query size	**Percentage of time spent**		
	10 kbase	**100kbase**	**1Mbase**
Stage 1	86.5%	83.3%	85.3%
Stage 2	13.3%	16.6%	14.65%
Stage 3	0.2%	0.1%	0.05%

II. TYPES OF BLAST PROGRAM

Blastp : Compares an amino acid query sequence against a protein sequence database.

Blastn : Compares a nucleotide query sequence against a nucleotide sequence database.

Blastx : Compares a nucleotide query sequencetranslated in all reading frames against a protein sequence database.

Tblastn : Compares a protein query sequence against a nucleotide sequence database dynamically translated in all reading frames.

Tblastx : Compares the six-frame translations of a nucleotide query sequence against the six-frame translations of a nucleotide sequence database. Note that tblastx program cannot be used with the database on the BLAST webpage.

Although many types exist in BLAST, the problem is particularly execute for BLASTN, the BLAST variant used to compare DNA sequences, because each new genome sequenced from animals or higher plants produces between 108 and 10 0 bytes of new DNA sequence. So, it is necessary to improve the searching process in BLASTN particularly.

A. *The Genetic Code*

The rules by which the nucleotide sequence of a gene is translated into the amino acid sequence of the corresponding protein, the so-called genetic code. The sequence of nucleotides in the MRAN molecule, that acts as an intermediate is found to be read in serial order in groups of three. Each triplet of nucleotides, called a codon, species one amino acid (the basic unit of a protein, analogous to nucleotides in DNA). Since RNA is a linear polymer of four different nucleotides, there are 43=63 possible codon triplets (However, only 20 different amino acids are commonly found in proteins, so that most amino acids are specified by several codons. In addition, 3 codons (of the 64) specify the end of translation, and are called stop codons. The codons specifying beginning of translation is AVG and is also the codon for the amino acid methionine. The code has been highly conserved during evolution: with a few minor exceptions, it is the same in organisms as diverse as bacteria, plants and humans.

III. WORD—MATCHING ACCELERATOR ARCHITECTURE

The first stage of BLASTN is used to find "seeds" or word matches. A word match is a string of fixed length w (referred to as "w-mer") that occurs in both the query sequence and the database sequence. This word-matching stage design can be decomposed into three sub stages as shown in figure 1

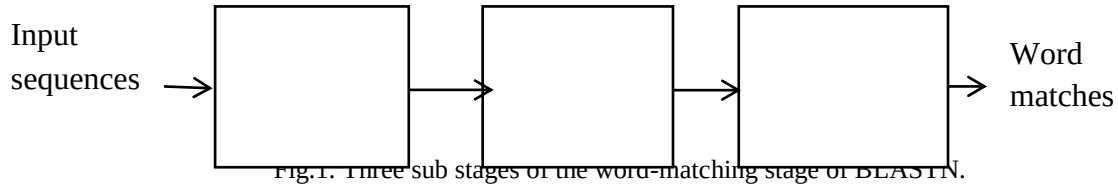


Fig.1. Three sub stages of the word-matching stage of BLASTN.

The first sub stage is a parallel bloom filter; the second sub stage is a false positive eliminator to examine the data passing the parallel bloom filters, and the last sub stage eliminates redundant matches.

A. *Parallel Bloom Filter Architecture*

Word matches could be computed using data structures such as hash tables or suffix trees. While bloom filters have been used to accelerate hash lookups in high-speed network packet filtering and routing, an solution to this filtrations problem is to use a bloom filter. A bloom filter is a simple space-efficient randomized hashing data structure suitable for quick membership tests on FPGA implementation.

A Bloom filter works in two steps.

1) Programming: For a given set I of n keys, I = {x1,….xn} the Bloom filter's programming process is described as follows. First of all, initialize the bit vector m with zeros, then, for each key xjEI, compute its k hash values. Hi (xj), $1 \le i \le k$, subsequently, set the bit vector to one according to the k hash values (i.e., BF $[h_i(x_j)]=1$ for all $1 \le i \le k$).
2) Querying: the querying process of the bloom filter works the same as its programming process. For a given key x, compute k hash values $h_i(x)$, $1 \le i \le k$. If any of the k-bits BF [hi(x)], $1 \le i \le k$ is zero otherwise, x is said to be a member of set I with contain probability.

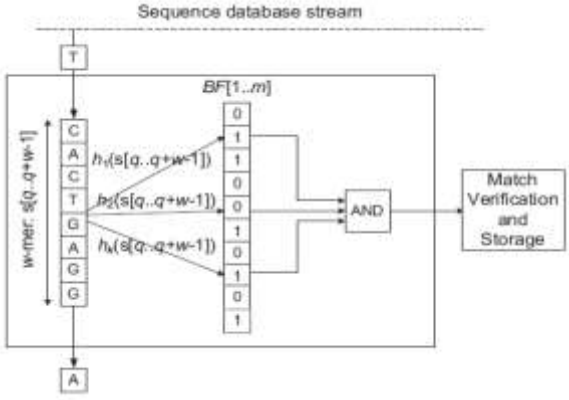


Fig.2 Conventional design for identifying w-mers using a Bloom filter.

The conventional design for the identification of w-mers using a bloom filter is considered. The bloom filter has been programmed by parsing the query sequence into overlapping substrings of length w in the preprocessing step. Although the conventional bloom filter architecture is efficient for membership test, its direct implementation is not suitable for the high performance design on an FPGA[13]. Current on-chip memory blocks only provide limited access at the same clock cycle. Pipelining or memory duplication is required for multiple memory requests.

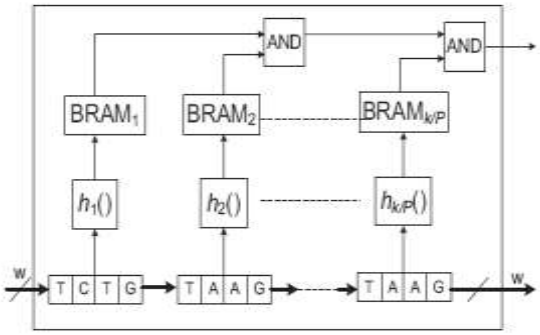


Fig.3. Parallel Bloom filter architecture with k/p hash functions

In contrast, a novel architecture for the bloom filter design is introduced to provide a computational efficiency, the parallel portioned bloom filter[12]. Three techniques are applied to improve the throughput compared to the conventional bloom filter architecture.

1) Partitioning: First partition the bloom filter vector into a number of smaller vectors, which are then queried by independent hash functions.

2) Pipelinining: The throughput of our design is further increased using a new pipelining technique.

3) Local Stalling: This mechanism is to guarantee all w-mers are tested by the bloom filter.

In order to accelerate the BLASTN process further, the bloom filter is modified using less hash functions. Generally, there are only four nucleotides such as Adenine, Cytosine, Guanine and Thymine by which sequences can be formed. Nucleotide sequences with fixed word length will be compared in the bloom filter. Word sizes may be different like two, three or four. In proposed system, word size is fixed to three nucleotides because triplets of nucleotides called codon will form one amino acid. Many codons represent the same amino acid. So the number of hash functions reduced from 60 to 20. Less hash functions leads to increase the speed of bloom filter and reduces the memory space.

B. *False Positive Eliminator Design*

One key feature for the bloom filter is that false-positive answers are possible. This is due to the fact that the hash function could hash two different keys into the same address with low probability[4].The second sub stage of the word-matching accelerator design is false-positive elimination. It includes two objectives:

1) Find all false-positive matches generated by the bloom filter;

2) Generate the corresponding position information in the query sequence for true-positive w-mers.

One solution for this sub-stage is to use a hash lookup table. The position information of each w-mers from the query sequence is stored in the hash table. A hash table with 1 million entries storing position information for a 100-k base query sequence requires at least 17m bits of memory space. So, the memory requirement is significantly greater[5]. Hash collisions and duplicate keys are two common problems for simple hashing strategies. Although it is difficult to find a perfect hash for all n keys, it might be easier to find a perfect hash function for a subset of keys, if the size of the subset is small enough. Bucket hashing work as follows:

1) Sort the query w-mers into different buckets according to their prefix.'

2) Find a simple hash function that is collision free for all w-mers in the same bucket. If it is not possible to find such a perfect hash function, uses the hash function with the minimum hash collisions.

3) Construct a quick lookup table (QLT) which stores the "collision-free" hash function for each bucket.

C. *Redundancy Eliminator Design*

In order to avoid repeated generation of the same sequence alignment during the ungapped extension stage of BLASTN, it uses a redundancy filter to eliminate w-mers that lead to the same ungapped extension range. Each w-mer is represented by an ordered pair (q_j, d_k), where q_j and d_k are indices of the query and database sequence, respectively. The diagonal of this w-mer is defined as $D=q_j-d_k$. Redundancy matches are eliminated by examining their diagonals. In NCBI BLASTN, it also used the feedback from the ungapped extension stage to eliminate redundancy matches. In contrast this design is less stringent. Here, only "true overlapping" match w-mers are eliminated (ie) if two consecutive matches are eliminated. The same diagonal and they have can overlapping part, the latter is discarded as a redundant match. The non-overlapping diagonal will be updated, once a non-overlapping match is found.

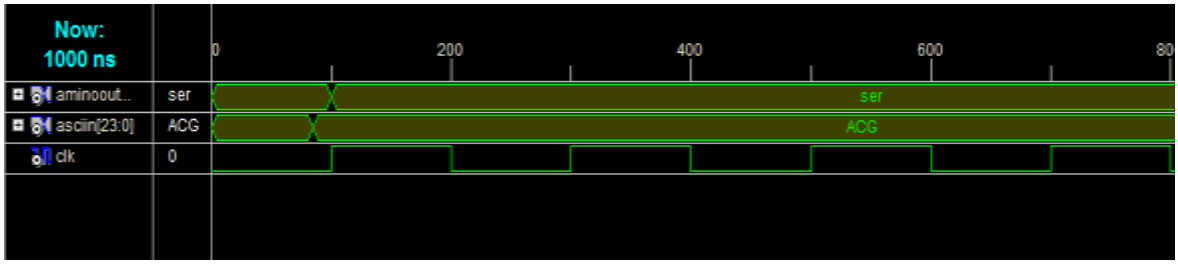
IV. RESULTS AND DISCUSSIONS

The word –matching stage accelerator has been implemented using Verilog HDL. This FPGA implementation is compared with the NCBI BLASTN, running on a general purpose workstation. The parameters for both the FPGA and the software program are set to the same default values. The mouse genome is chosen as the database sequence. Query sequences of different sizes have been randomly chosen from the human genome (10 kbase, 50 kbase, and 100 kbase). Hereafter the word matching stage will be integrated into DRC coprocessor system. A large value of off-chip data can be stored using the DRC system's memory, which consists of up to 8 GB of DDR2 SDRAM with a maximum bandwidth 3.2 GB/s and 512 MB of low latency RAM with a maximum bandwidth of 1.4 GB/s.

In order to quantify the performance improvements of our word-matching accelerator, several test to be designed to simulate possible large-scale DNA sequence comparisons. With the help of newer FPGA technologies, better performances can be expected with the same architecture. As far as we know, the latest virtex-7 FPGA chip can provide up to 85 MBit of block RAMs. This indicates that it could support much longer query sequences or construct the hash lookup tables using the on-chip memories to give a faster access speed.

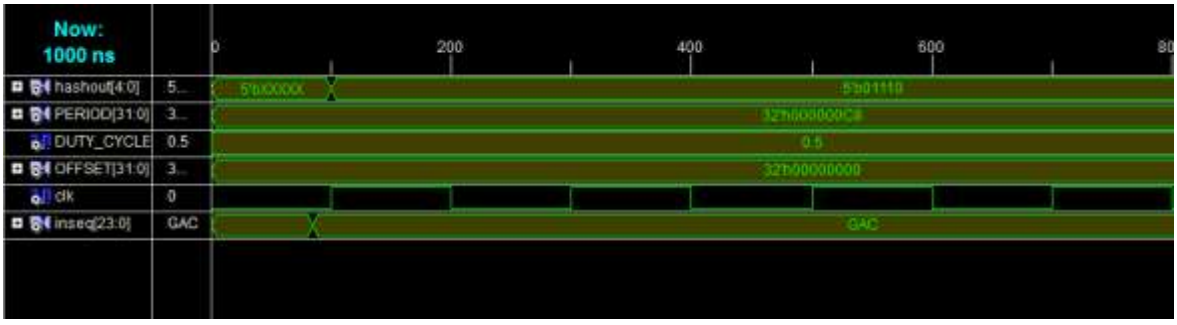
A.*Genetic codes of amino acids*

Three nucleotides together (i.e a codon) will form one aminoacid. After checking the match pair of codons , amino acids are compared because similarities can be found by amino acids only. Here 'asciiin' is the input which contains ASCII value of codons and 'aminoout' is the output which has genetic code of the respective codon . This process is simulated in Xilinx environment.



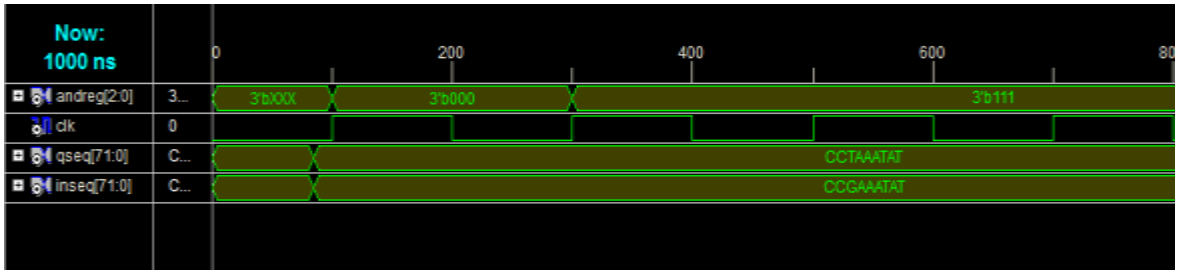
B.*Hash function of genetic code*

The hash table is a datastructure. It is used to map codons to their associative values.Here 'inseq' is the input of codon and 'hashout' is the output which has the hash function of the corresponding codon.

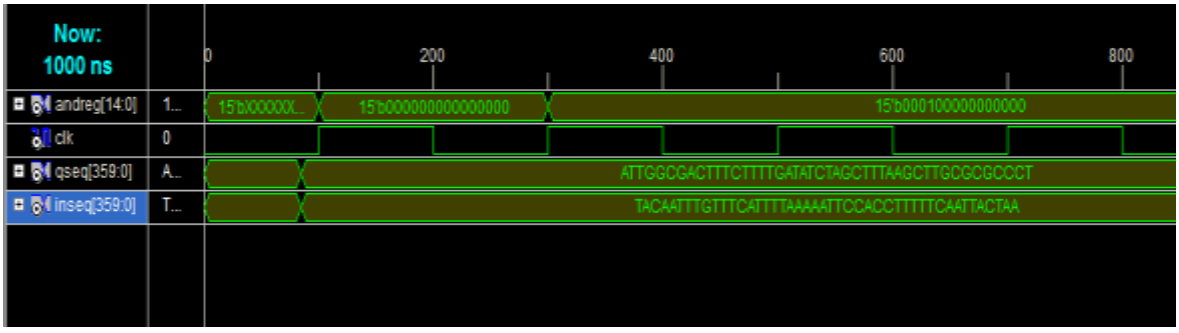


C.*Bloom Filter Design*

The bloom filter compares the query sequence and the database sequence and gives the output as 1 when both words are equal.

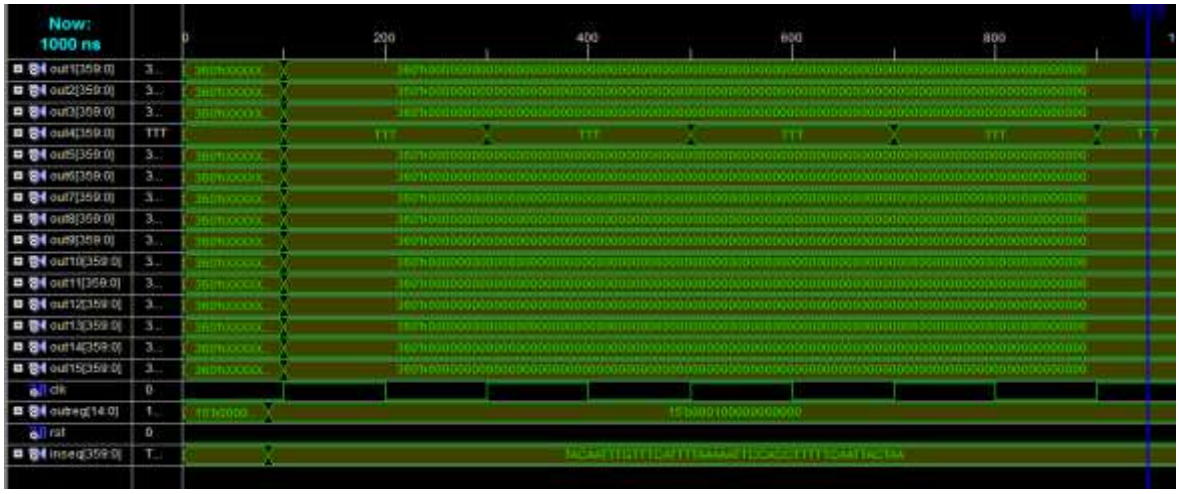


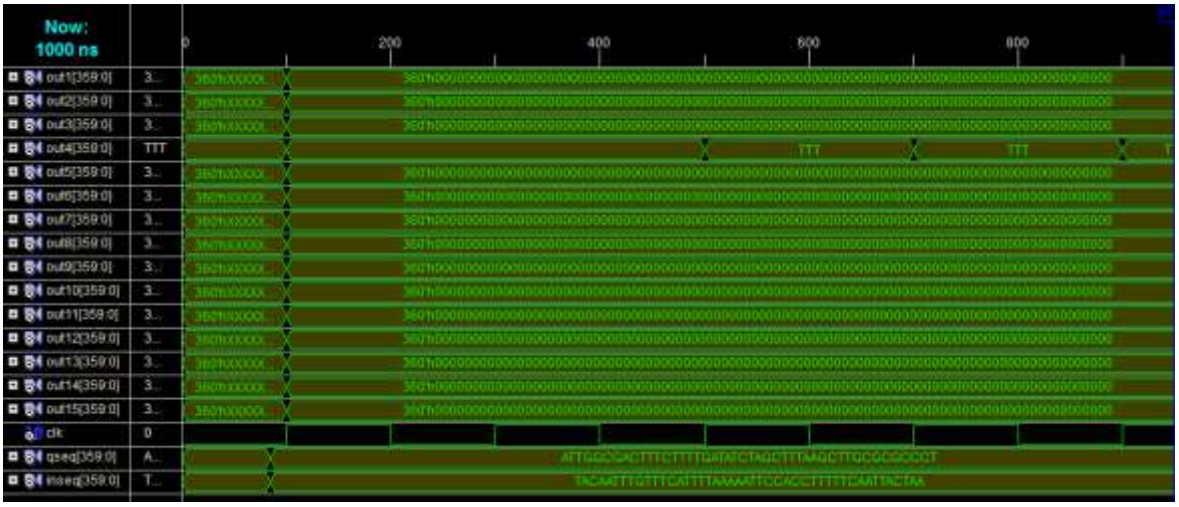
By increasing the input size (i.e., with 15 codons), the output for the bloom filter is given below



D.*Blast operation*

It uses the output of bloom filter and the input sequence to get the output of best possible matches.





By using this output of bloom filter and the BLAST operation , the word matching stage can be done efficienty by comparing the both input sequences and the output of best possible word matches.

V. CONCLUSION

In this paper, an FPGA based reconfigurable architecture is presented to accelerate the word matching stage of BLASTN, which is a bio sequence search tool of high importance to bioinformatics research. This design consists of three sub stages, a parallel bloom filter, an off-chip hash table, and a match redundancy eliminator. Different techniques are applied to optimize the performance of each sub stage. The comparison of the performance of this

word-matching accelerator to that of NCBI BLASTN shows a speedup around one order of magnitude with only modest resource utilization.

REFERENCES

[1] Iulian Moraru, David G. Andersen (2011), 'Exact Pattern Matching with Feed-Forward Bloom Filters' , ACM Journal of Experimental
[2] Algorithms, Vol.16, No.16, Article 2.1
[3] Jeremy D. Buhler, Joseph M. Lancaster, Arpith C. Jacob, and Roger D. Chamberlain, (2007), 'Mercury BLASTN: Faster DNA Sequence
[4] Comparison Using a Streaming Hardware Architecture', in Proc. of Reconfigurable Systems Summer Inst., Jul, pp.1-7.
[5] Jin H. Park YunfeiQiu Martin C. Herbordt (2010), 'CAAD BLASTn: Accelerated NCBI BLASTn with FPGA Prefiltering',Department of
[6] Electrical and Computer Engineering, Boston University, Boston, U.S.A
[7] KetilMalde and Bryan O'Sullivan (2009), 'Using Bloom Filters for Large Scale GeneSequence Analysis in Haskell', PADL 2009, LNCS
[8] 5418, pp. 183–194.
[9] Martin C. Herbordt, Josh Model, Bharat Sukhwani, YongfengGu and Tom Van Court (2007), 'Single Pass Streaming BLAST on FPGAs',
[10] Parallel Comput. November 33(10-11): 741–756.
[11] NCBI Programming with BLAST: http://www.ncbi.nlm.nih.gov/BLAST/
[12] Nourani.M and Katta.P (2007), 'Bloom Filter Accelerator for String Matching',IEEE, Center for Integrated Circuits & Systems,The
[13] University of Texas at Dallas.
[14] Praveen Krishnamurthy, Jeremy Buhler, Roger Chamberlain, Mark Franklin,Kwame Gyang, Arpith Jacob and Joseph Lancaster (2007),
[15] 'Biosequence similarity search on the Mercury System', Journal of vlsi signal processing, springer science + business media, LLC.
[16] Siddhartha Datta, ParagBeeraka , Ron Sass (2009),' RC-BLASTn: Implementation and Evaluation of the BLASTn Scan Function', 17th
[17] IEEE Symposium on Field Programmable Custom Computing Machines.
[18] XinyuGuo, Hong Wang and Vijay Devabhaktuni (2012), 'A Systolic Array-Based FPGA Parallel Architecture for the BLAST Algorithm',
[19] International Scholarly Research Network ISRN Bioinformatics Volume 2012, Article ID 195658, 11 pages.