Research Article **ISSN: XXXX – XXXX**

International Journal of Engineering and Scientific Research

Journal home page: www.ijesr.info

AN EFFICIENT RETOUCHED BLOOM FILTER BASED WORD-MATCHING STAGE OF BLASTN

R. Valli Suseela*[1]

*[1]Department of Electronics and Communication Engineering, Rajaas Engineering College, Tamilnadu, India.

ABSTRACT

BLAST is one of the most popular sequence analysis tools used by molecular biologists. It is designed to efficiently find similar regions between two sequences that have biological significance. However, because the size of genomic databases is growing rapidly, the computation time of BLAST, when performing a complete genomic database search, is continuously increasing. In this paper, we present a new approach for genomic sequence database scanning utilizing retouched bloom filter based FPGA architecture. An RBF is an extension that makes the Bloom filter more flexible by permitting the removal of false positives, at the expense of introducing false negatives, and that allows a controlled trade-off between the two. We further provide some simple heuristics that decrease the false positive rate more than the corresponding increase in the false negative rate, when creating RBFs. The RBF algorithms require space that is at most a small constant multiple of the Bloom filters vector size. Compared to the creation of a standard Bloom filter, the RBF algorithms also incur additional processing costs related to key removal. These costs are a constant multiple of a number of RBF parameters, such as the number of hash functions and the number of false positives to remove. These heuristics are more effective than the ones we have presented in prior work.

KEY WORDS

Retouched Bloom filter, Genomic sequence and Database search, w-mer scheduler, Hash table with signature analysis module and FPGA based word matching stage.

Author of correspondence:

R. Valli Suseela,
Department of Electronics and Communication Engineering,
Rajaas Engineering College,
Tamilnadu, India.
Email: r.vallisuseela@gmail.com.

INTRODUCTION[1]

Scanning genomic sequence databases is a common and often 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 of 1.5 to 2[1]. The aim of a scan operation is to find similarities between the query sequence and a particular genome sequence[1],

which might indicate similar functionality from a biological point of view. Dynamic programming-based alignment algorithms can guarantee to find all important similarities. How-ever, as the search space is the product of the two sequences, which could be several billion bases in size; it is generally not feasible to use a direct implementation. One frequently used approach to speed up this time-consuming operation is to use heuristics in the search algorithm. One of the most widely used sequence analysis tools to use heuristics is the basic local alignment search tool (BLAST)[2]. Although BLAST's algorithms are highly optimized for similarity search[1] the ever growing databases outpace the speed improvements that BLAST can provide on a general purpose PC.

BLASTN, a version of BLAST specifically designed for DNA sequence searches, consists of a three stage pipeline.

Stage-1

Word-Matching detect seeds (short exact matches of a certain length between the query sequence and the subject sequence), the inputs to this stage are strings of DNA bases, which typically uses the alphabet {A, C, G, T}.

Stage-2

Ungapped Extension extends each seed in both directions allowing substitutions only and outputs the resulting high-scoring segment pairs (HSPs). An HSP[3] indicates two sequence fragments with equal length whose alignment[2] score meets or exceeds a empirically set threshold (or cutoff score).

Stage-3

Gapped Extension uses the Smith-Waterman dynamic programming algorithm to extend the HSPs allowing insertions and deletions.

The basic idea underlying a BLASTN search is filtration. Although each stage in the BLASTN pipeline is becoming more sophisticated, the exponential increase in the volume of data makes it important that measures are taken to reduce the amount of data that needs to be processed. Filtration discards irrelevant fractions as early as possible, thus reducing the overall computation time. Analysis of the various stages of the BLASTN pipeline reveals that the word-matching stage is the most time-consuming part. Therefore, accelerating the computation of this stage will have the greatest effect on the overall performance. In this paper, we propose a computationally efficient architecture to accelerate the data processing of the word- matching stage based on field programmable gate arrays (FPGA). FPGAs are suitable candidate platforms for high-performance computation due to their fine-grained parallelism and pipelining capabilities.

RELATED WORK

There have been several approaches to accelerate bio sequence similarity searches. Some of these approaches use special hardware, while others attempt to solve this problem in software using better algorithms or heuristics. Hybrid approaches employ both the general purpose computer and specialized hardware. Mega BLAST is used by the National Center for Biotechnology Information (NCBI) as a faster alternative to BLASTN[4]. It achieves a faster processing speed by sacrificing substantial sensitivity. BLAST can quickly find alignments in sequences of high similarity. It indexes the entire database[7] offline before being used in a search. By eliminating the need to scan the database, it achieves more than an order of magnitude speedup comparing to BLASTN. However, a tradeoff is made between the processing speed and the sensitivity. Pattern Hunter[4] uses a spaced seed model to achieve faster processing speed and higher sensitivity; Pattern Hunter II[5] implements the optimized multiple seeds scheme to further increase the sensitivity. The spaced seed model is designed to gain profits based on the following constraints: an appropriately chosen model has a significantly higher probability of having at least one hit in a homologous region, compared to BLAST's consecutive seed model, while having a lower expected number of hits. If the spaced seed is not properly designed, there will be too many random hits to slow the subsequent computation.

However, to find an optimal seed of given weight and length is NP-hard. FPGAs can provide outstanding performance on parallel data processing, which make them a good option for algorithm acceleration. A small number of designs to speed up

BLAST's performance on FPGA devices have been presented. RC-BLAST[6] is an early implementation of BLAST. It first profiles the application to identify the compute-intensive segments. TUC-BLAST accelerates DNA searches for small query sequences (1000 characters) regardless of the database size[7]. It achieves a significant performance improvement compared to the BLAST software, but its performance for large query sequences is not clear. Indexes both the query sequence and the database sequence based on seed of length *w*.

Afterwards, the neighborhood information to conduct the extension from off chip Flash memory is extracted. Although Flash memory can provide a large space to store the indexed information, its bandwidth can become the performance bottleneck when throughput rate is the first priority[8]. Mercury[9] accelerates BLASTN's word-matching stage computation by applying parallel Bloom filters. In addition, the Mercury system also provides an efficient near-perfect hashing[10] strategy to eliminate false-positive answers. As BLASTN's word-matching stage is to detect exact matches between two sequences, the Bloom filter architecture is a proper choice to achieve this goal. In this paper, we also accelerate the word-matching stage computation using the Bloom filter architecture. Our approach differs from the Mercury[9] approach in that we apply a partitioned Bloom filter to provide better computational efficiency, and a bucket hashing[14] data structure to ease off-chip hash table accesses.

The Bloom[11] filter architecture has been used in a number of application fields (e.g., pattern matching). Various Bloom filter architectures have been proposed to achieve different functionalities. Dharmapurikar and Lockwood present a multipattern matching algorithm using multiple parallel Bloom filters on an FPGA. Each Bloom filter detects strings of a unique length[12]. Nourani and Katta[13] combine the Bloom filter and the parallel hash engines to achieve higher throughput. The feed forward Bloom filter applies a second bit array to reduce scan time and memory requirement for large number of patterns with relatively few matches.

IMPLEMENTATION PLATFORM

We have chosen the DRC coprocessor system as our target experiment platform. Acclaim is the third generation of DRC coprocessors. It is a high-performance computing system for processing-intensive applications, consisting of three Hyper Transport bus and six memory interfaces to the user's logic design. Application images are stored in Flash memory, and are used to configure the FPGA at power-on (Figure No.1).

WORD-MATCHING ACCELERATOR ARCHITECTURE

The first stage of BLASTN is used to find "seeds" or word matches. A word match is a string[12] of fixed length *w* (referred to as "*w*-mer") that occurs in both the query sequence and the database sequence. Using the alphabet {*A*, *C*, *G*, *T*}, NCBI BLASTN reduces storage and I/O bandwidth by storing the database using only 2 bits per letter (or base). The default *w*-mer length for a nucleotide search is set to 11. The word-matching stage implementation of NCBI BLASTN first examines *w*-mers on a byte boundary (i.e., 8-mers). Subsequently, exact 8-mer matches are extended in both directions to find possible 11-mer matches. If two matching 11-mers occur in close proximity, they are likely to generate the same HSPs. NCBI BLASTN therefore implements a redundancy eliminator to avoid repetitive inspections on the same segment in later stages. Our FPGA-based accelerator design for BLASTN does not follow exactly the same working mechanism presented in the NCBI BLASTN software. Instead, we have chosen FPGA favorable algorithms to achieve the same functionality. Our word-matching stage design can be decomposed into three sub stages, as shown in Figure No.2.

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. The sub stage composition is similar to that of Mercury[9], but the detailed architecture is different.

PARALLEL BLOOM FILTER ARCHITECTURE[14]

The word-matching stage aims to find good alignments containing short exact matches between a query sequence and a database sequence. Such matches could be computed using data structures such as hash tables or suffix trees. An alternative solution to this filtration problem is to use a Bloom filter. A Bloom filter is defined by a bit-vector of length *m*, denoted as *BF* [1, …, *m*]. A family of *k* hash functions h_i: $S \rightarrow A$, $1 \leq i \leq k$, is associated to the Bloom filter, where *S* is the key space and $A = \{1, \ldots, m\}$ is the address space. A Bloom filter is a simple space-efficient randomized hashing[10] data structure suitable for quick membership tests on FPGA implementations.

A Bloom filter works in two steps

1. **Programming**
   For a given I of keys, set n $I=\{x_1,\ldots x_n\} \subseteq S$, the filter programming process described as follows. First of all initialize the bit vector m with zeros for each key xj compute its hash values $h_i(x_j)$, $1 \leq i \leq k$, subsequently set the bit vector to one according to the *k* hash values (i.e., $BF[h_i(x_j)] := 1$ for all $1 \leq i \leq k$).
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 $h_i(x)$, $1 \leq i \leq k$. If any of the *k* bits $BF[h_i(x)]$, $1 \leq i \leq k$, is zero, then $x \notin I$, otherwise, *x* is said to be a member of set *I* with a certain probability.
   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. The Bloom filter only produces false-positive results but never false-negative answers to the query. The false-positive probability (FPP) of a Bloom filter is given by,

$$\mathbf{FPP = (1 - (1- 1/m)^{kn})^{k} = (1 - e^{-kn/m})^{k}} \quad \text{----------(1)}$$

In our previous work, we have implemented a 4 × 4 parallel partitioned Bloom filter to test the computation efficiency introduced by the partitioned architecture. In this paper, we further analyze the influences of different architecture configurations on the partitioned Bloom filter. Based on the query sequence and database sequence, we implement an 8 × 2 parallel Bloom filter architecture, which theoretically doubles the throughput comparing to the 4 × 4 architecture under zero-match condition, to gain better performance. Our design takes advantage of the fact that mismatches appear far more frequently than matches in the BLASTN word-matching stage and a match key has much tighter requirements than mismatches (once a key fails in any of the *k* hash queries, it will be defined as a mismatch, in contrast, a match key should pass all hash queries). The computation efficiency will be compromised, if a single key was sent to all hash functions for membership testing, especially under low match rate conditions. Thus, our idea is to divide the *k* hash functions into different groups, with each group used for a different hash query. We apply three techniques to improve the throughput compared to the conventional Bloom filter architecture.

a. **Partitioning**
   We first partition the Bloom filter vector into a number of smaller vectors, which are then queried by independent hash functions.
b. **Pipelining**
   We further increase the throughput of our design using a new pipelining technique.
c. **Local stalling**
   We use a local stalling mechanism to guarantee all *w*-mers are tested by the Bloom filter. Our basic building block is a pipelined partitioned Bloom filter with k/P hash functions, denoted as PPBF (k/P), where *P* is the degree of parallelism. In each clock cycle, it can support k/P different hash queries. The hash functions used in the PPBF block are chosen from the H_3 family, which can be efficiently implemented in hardware (Figure No.3 and 4). Suppose the input bit string *X* with *b* bits is represented as *X* = <

$x_1, x_2, ..., x_b>$.We calculate the i -th hash function over X, $h_i(X)$ as

$$h_i(X) = (d_{i1} \cdot x_1) \oplus (d_{i2} \cdot x_2) \oplus \cdots \oplus (d_{ib} \cdot x_b) \quad \text{---------(2)}$$

MODULES FLOW ARCHITECTURE

A. The rule for the w-mer scheduler

1. If w-mer buffer$_i$ is empty, PPBF$_i$ can process data from its nearest w-mer buffer;
2. If only one w-mer is non-empty, all PPBFs can process data from this buffer;
3. If all w-mer buffers are empty, update all w-mer buffers simultaneously.

Buffer cond is a control signal that informs the w-mer scheduler about the empty buffers.

B. False-Positive Eliminator Design

The second substage of our word-matching accelerator design is false-positive elimination, which includes two objectives:

1. Find all false-positive matches generated by the Bloom filter;
2. Get the corresponding position information in the query sequence for true-positive *w*-mers.

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 uses the feedback from the ungapped extension stage to eliminate redundancy matches. We only eliminate "true overlapping" match *w*-mers, i.e., if two consecutive matches share the same diagonal and they have an overlapping part, we discard the latter as a redundant match. The non-overlapping diagonal will be updated, once a non- overlapping match is found.

PERFORMANCE ANALYSIS

The word-matching stage accelerator has been implemented using Verilog HDL and integrated into the DRC coprocessor system. In the DRC system, a Xilinx Virtex-5 LX330 FPGA chip is available for the user application. A large volume 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 each clock cycle, the parallel Bloom filter[10] can receive up to 16 new *w*-mers to do the membership examination from local buffers. The final design consumes about 47% of the slice registers, 50% of the LUTs, and 85% of the on-chip memory resource (about 2 Mbits for the *m*-bit vector in the Bloom filter design). In order to quantify the performance improvements of our word-matching accelerator, we have designed several tests to simulate possible large-scale DNA sequence comparisons.

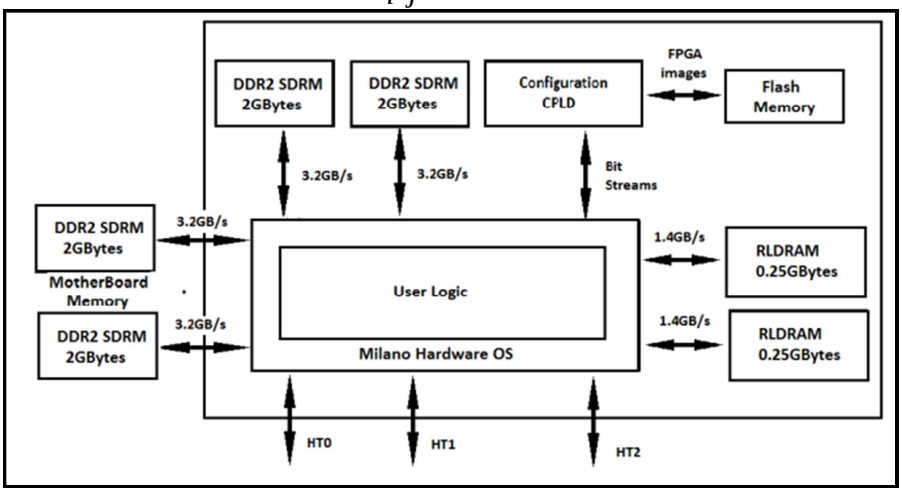

Figure No.1: DRC coprocessor diagram

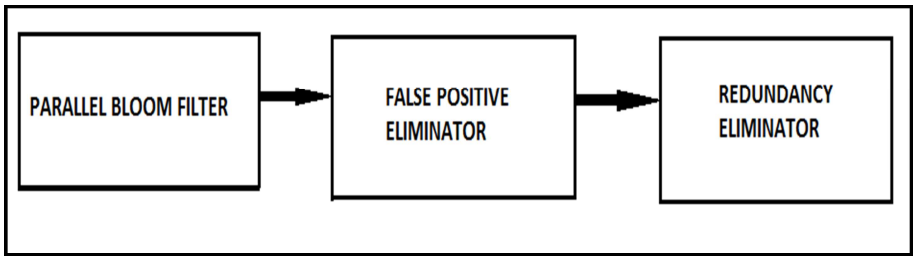

Figure No.2: Word Matching Stage

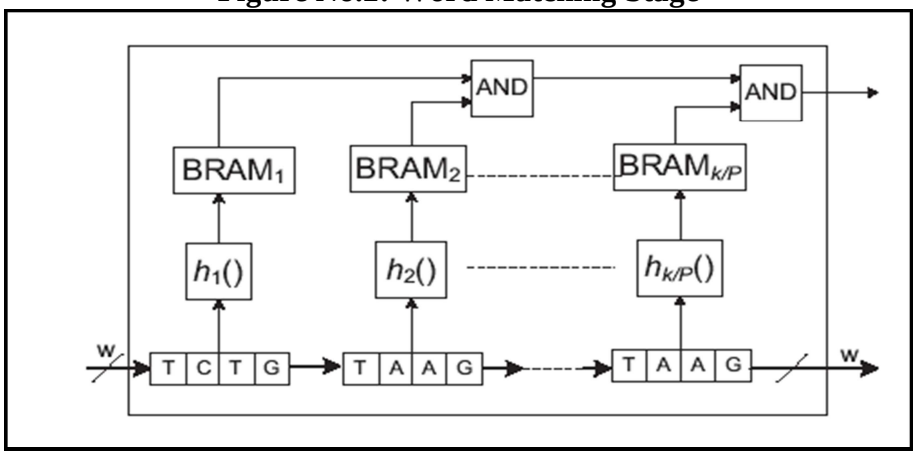

Figure No.3: PPBF architecture with *k*/*P* hash functions

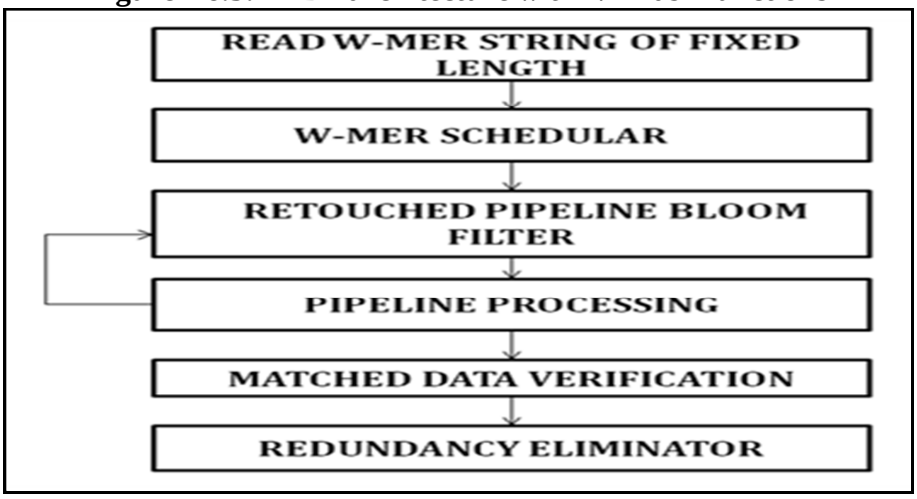

Figure No.4: Functional Data Flow Diagram

CONCLUSION

In this paper, I have presented an FPGA[7]-based reconfigurable architecture[8] to accelerate the word-matching stage of BLASTN, which is a bio-sequence search tool of high importance to Bioinformatics research. Our 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 our word-matching accelerator to that of NCBI BLASTN shows a speedup around one order of magnitude with only modest resource utilization. As FPGA-based designs exhibit high performance for parallel computing and fine-grained pipelining, we can expect obvious performance improvements of other applications in Bioinformatics. Therefore, we are also planning to design architecture for Stage 2 of the BLASTN pipeline (ungapped extension) in order to further improve the overall application performance.

ACKNOWLEDGEMENT

I would like to express my sincere and heart full thanks to my project internal guide Mr. C. Annapalagan, M.E., Assistant Professor, Head of the Department of Electronics and Communication Engineering, for his full support both mentally and technically by encouraging me during the course of work.

CONFLICT OF INTEREST

We declare that we have no conflict of interest.

BIBLIOGRAPHY

1. Krishnamurthy P, Buhler J, Chamberlain R, Franklin M, Gyang K, Jacob and Lancaster J. "Biosequence similarity search on the mercury system," *J. VLSI Signal Process. Syst*., 49(1), 2007, 101-121.
2. Zhang Z, Schwartz S, Wanger L and Miller W. "A greedy algorithm for aligning DNA sequences," *J. Comput. Biol.*, 7(1-2), 2000, 203-214.
3. Kent W J. "BLAT–the BLAST-like alignment tool," *Genome Res.*, 12, 2002, 656-664.
4. Ma B, Tromp J and Li M. "Patternhunter: Faster and more sensitive homology search," *Bioinformatics*, 18(3), 2002, 440-445.
5. Li M, Ma B, Kisman D and Tromp J. "Patternhunter II: Highly sensitive and fast homology search," *J. Bioinf. Comput. Biol*., 2(3), 2004, 417-439.
6. Muriki K, Underwood K D and Sass R. "RC-BLAST: Toward aportable, cost-effective open source hardware implementation," *in Proc. 19th Int. Parallel Distrib. Process. Symp*., 8, 2005, 1-8.
7. Sotiriades E, Kozanitis C and Dollas A. "FPGA based architecture for DNA sequence comparison and database search," *in Proc. 20th Int. Parallel Distrib. Process. Symp.*, 20th, 2006, 8.
8. Lavenier D, Georges G and Liu X. "A reconfigurable index FLASH memory tailored to seed-based genomic sequence comparison algorithms," *JVLSI Signal Process. Syst., Special Issue Comput. Archit. Accelerat. Bioinf. Algorithms*, 48(3), 2007, 255-269.
9. Buhler J, Lancaster J, Jacob A and Chamberlain R. "Mercury BLASTN: Faster DNA sequence comparison using a streaming architecture", *Roc.Reconfig. Syst. Summer Inst.*, 14, 2007, 1-7.
10. Ramakrishna M, Fu E and Bahcekapili E. "Efficient hardware hashing functions for high performance computers," *IEEE Trans. Comput.*, 46(12), 1997, 1378-1381.
11. Bloom B. "Space/time trade-offs in hash coding with allowable errors," *Commun. ACM*, 13(7), 1970, 422-426.
12. Dharmapurikar S and Lockwood J. "Fast and scalable pattern matching for network intrusion detection systems," *IEEE J. Sel. Areas Commun*., 24(10), 2006, 1781-1792.
13. Nourani M and Katta P. "Bloom filter accelerator for string matching," *in Proc. 6th Int. Conf. Comput. Commun. Netw.*, 6th, 2007, 185-190.
14. Moraru and Andersen D G. "Exact pattern matching with feed forward Bloom filter," *in Proc. Workshop Algorithm Eng. Experim.*, 2011, 1-12.

Please cite this article in press as: R. Valli Suseela. An Efficient Retouched Bloom Filter Based Word-Matching Stage of Blastn *International Journal of Engineering and Robot Technology*, 1(1), 2014, 25 - 31.