INTERNATIONAL JOURNAL OF PURE AND APPLIED RESEARCH IN ENGINEERING AND TECHNOLOGY

A PATH FOR HORIZING YOUR INNOVATIVE WORK

HIGH SPEED RECONFIGURABLE ACCELERATOR FOR WORD MATCHING STAGE OF BLASTN

DHARSANA P

Assistant Professor, Electronics And Communication Engg. Department, Apollo Engineering College, Chennai, Tamilnadu, India

Accepted Date: 22/11/2014; Published Date: 01/12/2014

Abstract: BLAST (Basic Local Alignment Search Tool) 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**.** There is a clear need to accelerate this process, In this paper, we present a new approach for genomic sequence database by using hash function to accelerate this word matching process. In order to derive an efficient structure for BLASTN, we propose a high speed reconfigurable architecture to accelerate the computation of the word-matching stage. It show the output 1 the word is matching it shows0 the word is not matched. The experimental results show that the FPGA implementation achieves a speedup around one order of magnitude compared to the NCBI BLASTN.

Keywords: Bloom filter, BLASTN, genomic sequence, reconfigurable accelerator, word matching.

Corresponding Author: MR. P. DHARSANA

Access Online On:

www.ijpret.com

How to Cite This Article:

Dharsana P, IJPRET, 2014; Volume 3 (4): 149-156

PAPER-QR CODE

INTRODUCTION

The aim of a scan operation is to find similarities between the query sequence and a particular genome sequence, which might indicate similar functionality from a biological point of view. 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. 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. DNA sequence comparison is a computationally intensive problem, known widely since the competition for human DNA decryption.

RELATED WORKS

There are several methods to accelerate the process of bio sequence similarity search some of the approach use hard ware while other approaches using software algorithm. Hybrid approaches employed both general purpose computer and specialized hardware. Mega BLAST is used by the National Center for Biotechnology Information (NCBI) as a faster alternative to BLASTN. It achieves a faster processing speed by sacrificing substantial sensitivity. 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 uses a spaced seed model to achieve faster processing speed and higher sensitivity; Pattern Hunter implements the optimized multiple seeds scheme to further increase the sensitivity.

FPGA provide outstanding performance on parallel data processing, which make them a good option for algorithm acceleration. A small number of designs to speed BLAST’s performance on FPGA devices have been presented. RC-BLAST 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. It achieves a significant performance improvement compared to the BLAST software, but its performance for large query sequences is not clear

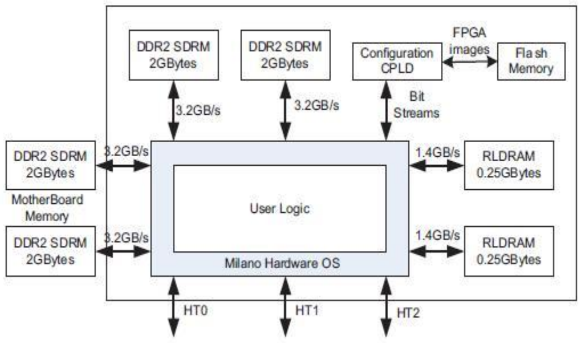


Fig 1: DRC coprocessor diagram

We have chosen the DRC coprocessor system as our target experiment platform. Accelium 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.

PROPOSED SYSTEM

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. 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 . 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.

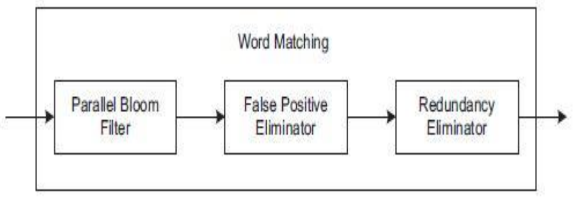


Fig 2: FPGA based accelerator for word matching stage of blastn.

Parallel bloom filter: Bloom filters use a randomized technique to test membership queries on a set of strings. Given a string, the Bloom filter computes hash functions on it producing hash values ranging from 1 to m. It then sets k bits in a m bit long vector at the addresses corresponding to the hash values. The same procedure is repeated for all the members of the set. This process is called "programming" of the filter. The Bloom filter generates hash values using the same hash functions it used to program the filter. Bloom filter is a space-efficient probabilistic data structure that is used to test whether an element is a member of a set Bloom filters use a randomized technique to test membership queries on a set of strings. Given a string, the Bloom filter computes hash functions on it producing hash values ranging from 1 to m. A Bloom filter is a simple space-efficient randomized data structure for representing a set in order to support membership queries. Although Bloom filters allow false positives, the space savings often outweigh this drawback.

The Bloom filter and its many variations have proven increasingly important for many applications (see, for instance, the survey As just a partial listing of examples, counting Bloom filters allow deletion The Bloom filter is a space-efficient probabilistic data structure that supports set membership queries. The data structure was conceived by Burton H. Bloomin . The structure offers a compact probabilistic way to represent a set that can result in false positives (claiming an element to be part of the set when it was not inserted), but never in false negatives (reporting an inserted element to be absent from the set). This makes Bloom filters useful for many different kinds of tasks that involve lists and sets.

False positive eliminator: The second substage of our word-matching accelerator design is false-positive elimination, which includes twoobjectives: find all false-positive matches generated by the Bloom filter; get the corresponding position information in the query sequence for true-positive w-mers. One solution for this substage is to use a hash lookup table. The position information of each w-mer from the query sequence is stored in the hash table. A hash table with 1 million entries storing position information for a 100-kbase query sequence requires at least 17 Mbits of memory space (17 bits are needed to represent 100 k positions). It is clear that the memory required is significantly greater than that provided by the on-chip BRAMs. Thus, we store the hash table in an external SDRAM attached to the FPGA.

Redundancy eliminator: In order to avoid repeated generation of the same sequence alignment during the un gapped extension stage of BLASTN, it uses a redundancy filter to eliminate w-mers that lead to the same un gapped extension range. Each w-mer is represented by an ordered where q j and dk are indices of the query and database sequence, respectively. The diagonal of this w-mer is defined as 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. In contrast, our design less stringent. We only eliminate "true overlapping" match w-mers 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.Although our heuristic is less stringent than NCBI BLASTN' there is no significant influence on the overall performance.

A hash function maps bit strings of arbitrary finite length of bit into fixed length many-to-one mapping. hashing is done for indexing and locating items in databases because it is easier to find the hash value than the longer string. Hashing is also used in encryption. The hash function transforms the digital signature, then both the hash value and signature are sent to the receiver. The receiver uses the same hash function to generate the hash value and then compares it to that received value.

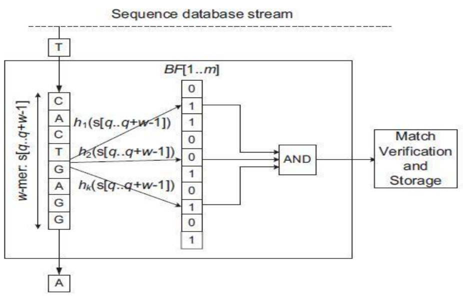


Fig3: Conventional design for identifying sequence database stream using a bloom filter

Clock frequency of the block RAMs to reduce memory consumption, but still requires four copies of the m-bit vector to process 16 hash queries at the same clock cycle. The limited on-chip memory becomes the bottleneck for the use of Bloom filter. In contrast, we introduced a novel architecture for the Bloom filter design to provide a better computational efficiency, the parallel partitioned Bloom filter. 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. The detailed analysis is presented in Section V. 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.

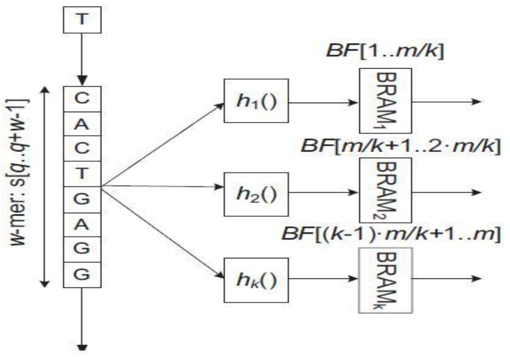


Fig 4: Partitioned bloom filter architecture using BRAM chip modules

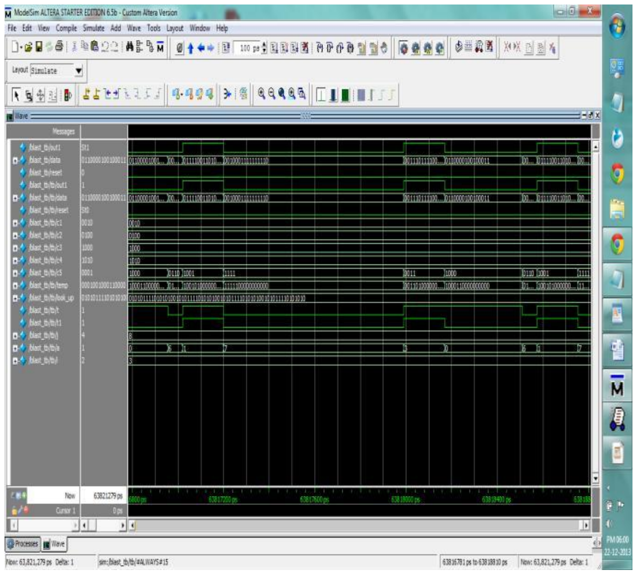
The computation efficiency will be compromised, if a single key was sent to all hash functions for membership testing, especially under low match rate conditions. Thus, our idea is to divide the k hash functions into different groups, with each group used for a different hash query. We apply three techniques to improve the throughput compared to the conventional Bloom filter architecture A perfect hash function maps a static set of n keys into a set of m integer numbers without collisions, where m is greater than or equal to n. If m is equal to n, the function is called minimal. a perfect hash function and a minimal perfect hash function (MPHF). Minimal perfect hash functions are widely used for memory efficient storage and fast retrieval of items from static sets, such as words in natural languages, reserved word sin programming languages or interactive systems.

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 can receive up to 16 new wmers 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 M bits 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. However, longer query segments introduce a new constraint on future Bloom filter design, which is not addressed in previous designs from the literature. In fact, the balance between the degrees of parallelism, the maximum query size each iteration and the computation time should be taken into account. One possible solution to moderate the off-chip memory access bottleneck is to use memory with a higher bandwidth provide multiple off-chip memory interfaces.

OUTPUT WAVEFORM



CONCLUSION

In this project work, FPGA-based reconfigurable architecture to accelerate the word matching stage of BLASTN .FPGA-based designs of bloom filter using hash function exhibit high performance , which will improve the performance of other applications in Bioinformatics 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.

REFERENCES

1. E. Sotiriades, C. Kozanitis, and A. Dollas, "FPGA based architecture for DNA sequence comparison and database search," in Proc. 20th Int. Parallel Distribute. Process. Symp., 2006, p. 8.

2. P. Karishnamurthy, J. Buhler, R. Chamberlain, M. Franklin, K. Gyang,A. Jacob, and J. Lancaster, "Bio sequence similarity search on the mercury system," J. VLSI Signal Process. Syst., vol. 49, no. 1, pp. 101– 121, 2007.

3. Z. Zhang, S. Schwartz, L. Wanger, and W. Miller, "A greedy algorithm for aligning DNA sequences," J. Comput. Biol., vol. 7, nos. 1–2, pp. 203–214, 2000.

4. M. Ramakrishna, E. Fu, and E. Bahcekapili, "Efficient hardware hashing functions for high performance computers," IEEE Trans. Computer., vol. 46, no. 12, pp. 1378–1381, Dec. 1997.