HBLast: An open-source FPGA library for DNA sequencing acceleration

Marzhan Bekbolat, Sabina Kairatova, Ayan Shymyrbay and Kizheppatt Vipin
Department of Electrical and Computer Engineering
Nazarbayev University
Astana, Kazakhstan
{marzhan.bekbolat,sabina.kairatova,ayan.shymyrbay,vipin.kizheppatt}@nu.edu.kz

***Abstract*—Biological sequence alignment algorithms work by identifying the degree of similarity between a newly discovered biological sequence with already known sequences. Exponential growth in the database and query sizes make pure software-based solutions inefficient in finding a solution within the time constraints. This calls for hardware acceleration-based solutions such as those based on field programmable gate arrays (FPGAs). In this work we present the design and implementation of the popular BLAST algorithm for nucleotide sequence alignment, called HBLast, targeting FPGA-based reconfigurable platforms. The implementation is supported by both on-premise (private) FPGA instances as well as cloud-based solutions. Results show that the platform can achieve high-performance which keeping low resource consumption. The platform is available as an open-source, enabling other researchers and small research centers to achieve cloud-based genomic sequence acceleration.**

***Index Terms*—FPGA, BLAST algorithm, Parallel architecture, Sequence alignment.**

I. Introduction

Biological sequence alignment algorithms are widely used in Bioinformatics and Computational biology (BCB), where an unknown DNA sequence is searched against sequences from a large database. The algorithm works by identifying the degree of similarity between a newly discovered biological sequence with already known sequences [1]. These algorithms can lead to early disease diagnosis, where the biological information of a newly discovered infectant (virus or bacteria) DNA sequence can be obtained by matching it with the most similar disease genes in the database [2]. However, sequence alignment algorithms are highly compute intensive [3]. This calls for powerful servers for running the software tools and will be beyond the capabilities of small hospitals and research centers. Even on high performance computers, often the time required by pure software implementations can vary from several minutes to hours [4].

In recent years, Field Programmable Gate Arrays (FPGAs) have been proposed as a platform for accelerating bioinformatic applications [1]. FPGA-based solutions have several advantages over pure software implementations for such compute-intensive applications. The hardware flexibility of FPGA architecture enables highly efficient operations at narrow bit-widths compared to fixed register size of traditional processors [5]. Secondly, FPGA power consumption is significantly lower (up to 4×) compared to CPUs [5]. Moreover, reprogrammability of FPGAs enables them to time-multiplex different accelerator applications or even portions of the same algorithm to fit in a smaller device. The reprogrammability also makes them attractive compared to other more expensive hardware implementation such as ASICs.

The efficiency of the search algorithm is essential, and probably as important as the hardware it runs on. There are various biological sequence alignment techniques developed over the past decades [6]. Basic Local Alignment Search Tool (BLAST) is one of the widely used heuristic methodologies, which delivers the best local alignment for large size of data sets. Due to its heuristic nature, BLAST is much faster than dynamic programming algorithms such as Needleman-Wunsch and Smith-Waterman algorithms [7]. Although BLAST has been proved to meet the performance and search sensitivity criteria, the improvements in DNA sequencing technologies rises new challenges for BLAST. Statistics imply that the number of genomic sequences is doubling almost every year, and therefore hardware acceleration is essential to meet the new requirements [7].

We propose an FPGA-based implementation of BLAST algorithm called HBlast, where the DNA search alignment is improved by parallel processing. This work is part of a larger project, which aims to develop a library of hardware accelerators targeting cloud-based FPGA instances. The accelerators will be provided in the public domain, which enables hospitals and research centers with limited computing facilities to achieve hardware acceleration by hosting these accelerators on the cloud infrastructure such as Amazon AWS FPGA instances [8]. Since the cloud billing model is based on usage and instance types, organizations can obtain their solution faster within the budget constraints. Accelerators could be also used in on-premises (private) FPGA instances since a communication infrastructure for interfacing the accelerators with standard PCIe and DRAM interfaces is also provided.

The main contributions of this work are

- Detailed architecture description of the FPGA-based implementation of the BLAST algorithm.
- An open-source implementation of the proposed algorithm targeting the Xilinx Virtex-7 and Amazon EC2 F1 FPGA instances.
- Characterization of the proposed platform in terms of FPGA resource consumption and performance.

The rest of this paper is organized as, Section 2 discusses the related work, Section 3 discusses the details of BLAST algorithm, Section 4 presents the proposed architecture and the implementation details, Section 5 provides the results of performance analysis, and Section 6 concludes the paper.

II. Related Works

Presently the most popular BLAST implementation is that of the National Center for Biotechnology Information (NCBI) [9], which runs on the NCBI servers. The high flexibility and scalability of software implementation and easy to use web-based interface make NCBI implementation very attractive. However, CPU based server technology is becoming insufficient in meeting the performance requirement due to the rapid growth in database size as well as query sequences. NCBI has proposed a number of solutions to improve the efficiency of software implementation such as the two-hit method and the Gapped BLAST [10]. In spite of enhancements, the software tools still takes long run-time. Hardware accelerators have high potential in this regard, since they can provide a high degree of bit-level parallelism. The proposed implementation is not an alternate for NCBI implementation, but try to complement it with support for hardware acceleration.

FPGAs are utilized in sequence comparison in Large Sequence Databanks since they significantly improve performance due to their parallel architecture. There are a number of proposed and applied implementations of BLAST algorithm targeting FPGAs [11]. Almost all of the implementations are based on one of the two searching strategies, the Word Position Record Based Search (WPRBS) and Systolic arrays [2]. The latter is a relatively new method and not a widely used for implementation of BLAST algorithm.

In WPRBS technique, only one word can be searched per clock, and consequently, only one hit can be found per clock [12]. Timing issue is also caused by shortage in WPRBS searching capacity due to limited number of memory ports in FPGAs. Examples of such architecture are Mercury BLASTn [13] and Mitrion [2]. In these two systems, storage tables are stored in the external memory SRAM that is attached to the FPGA, since the tables storing the words of long query sequences will require vast amount of space in internal memory RAM. Searching is performed by comparing one subject sequence from database to every word in the storage table simultaneously during one clock cycle. It is clear that performance of the system will be low since accessing external memory SRAM for getting subject sequence from database every time consumes significant amount of time.

By addressing the above timing issue, FPGA/FLASH [14] could achieve better results by minimizing the number of accesses to the external memory. This is done by enabling detection of several exact matches/hits simultaneously. This is due to the index structure of the database, where each word is associated with its position in the database and neighboring elements, hence, avoiding random accesses to database [2]. However, large memory space is needed to store index of the database as well as database itself. For database exponentially growing every year, this architecture will not be efficient in terms of storage capacity. In order to achieve better searching results, the architecture Multiengines BLASTn [15] was proposed. It is still based on WRPBS, but designed in such way that compares 64 subject sequences simultaneously due to its 64 identical computing units. As all of the above-mentioned implementations based on WRPBS, they suffer from timing and memory limitations. Further these architectures are not available as open source implementations preventing their wider adoption.

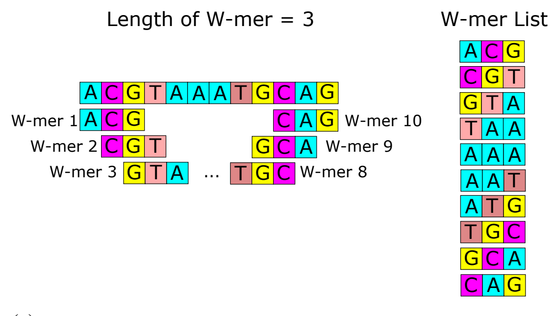
There are a few projects which are available as open source. RC-BLAST [3] and systolic array-based architecture [12] are two such implementations. Although, RC-BLAST is stated to outperform a modern's day general purpose PC computer, this implementation is mostly dependent on the database length, query length and the number of hits identified. For instance, this system's operation linearly depends on the size of the query, that is if query length increases, the FPGA system's performance rises. Also, RC-BLAST's performance drops if there are many hits found, which increases the frequency of memory access.

In another open source implementation [12], the researchers present a systolic array-based FPGA parallel architecture for BLAST algorithm. This architecture can detect more than one hit per clock cycle making it faster than WPRBS-based architectures. The speed advantage of this architecture is obtained by merging two adjacent hits into one and speed up the extension step. This implementation manages resource utilization better than Tree-BLAST [16], performs scanning faster than Mercury BLASTn [13], and deals with long sequences more effectively than Mercury BLASTp [17]. However, the architecture focuses only on the first two steps of BLAST algorithm, finding the hit and ungapped expansion.

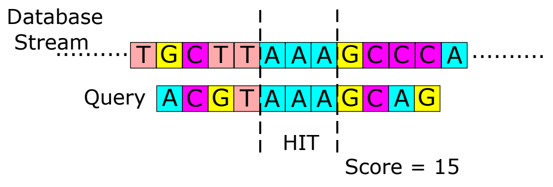
HBlast is a complete system-level implementation of BLAST targeting both on-premises as well as cloud-based acceleration. The implementation supports latest off-the-self FPGA boards as well as FPGA instances in the cloud. The support for cloud-based FPGA instances enables users dynamically scale the system based on their performance-cost constraints. The implementation can detect multiple word hits simultaneously due to the parallel implementation of the detection array. The query sequence is stored in the FPGA-internal memory and the database in the high-speed external DRAM memory. This minimizes the number of external memory accesses at the same time supports large database sequences.

III. Blast Algorithm

BLAST algorithm is used to search for similar parts of biological sequences in a database and a query [18]. Both the database and the query sequences are represented using strings of English alphabet with each letter representing a nucleobase such as Adenine or Thymine. There are multiple implementations of BLAST based on the type of data they process. DNA sequence search uses nucleotides with 4-letter



(a)



(b)

Fig. 1. (a) Finding W-mers from a query by creating substrings of size 3, (b) Example of exact match between a query and a database sequence

representation and protein search uses amino-acids with 20-letter representation. The inputs to BLAST are two sequences; a database, which consists of a large amount of classified data such as known viral DNAs, and a query, such as a piece of unknown DNA, which has to be compared with the database and find the similar parts. The outputs from the algorithm are the degree (score) of similarity of the aligned parts and their corresponding location in the sequences. Each matched pair between the database and the query is called a High Score Pair (HSP) and is of extreme importance for further biological computations. BLAST algorithm consists of 3 primary steps:

- *Pre-processing the query sequence*. The query is divided into several small portions/words.
- *Searching for hit*. Obtained small words of query are compared to the data in database to determine the exact match.
- *Expansion of comparison around hits*. Locations where the sequences coincide, the comparison is continued to both sides and HSP is calculated.

Details of each step is discussed in detail below.

1) **Pre-processing:** In this step, a query is divided into a list of substrings, as shown in Fig. 1 (a). These substrings are called W-mers of length W [15]. Let us assume we have a query DNA sequence ACGTAAARGCAG of length 12 and W-mers of length 3 [15]. Since the W-mers are contiguous substrings, there are in total 10 W-mers. The total number of W-mers in a querry can be calculated as

$$No.of\ Wmers = Query\ Length - Wmer\ Length + 1 \quad (1)$$

The substring ACG will be the first W-mer, CGT the second and so on.

2) **Searching for hits:** After the query is divided and the list of W-mers is generated, the search for exact match between W-mers and database is conducted. W-mers are chosen and randomly compared with the portions of the database. Every

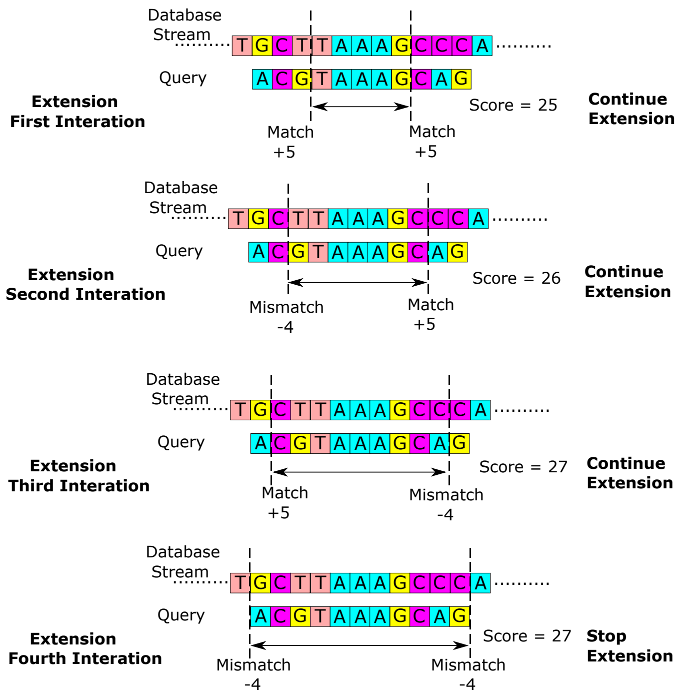


Fig. 2. Example of BLAST Algorithm's Third Step: Expansion of Comparison around Hits

exact match found is recorded as a "hit" and saved to process further in step 3 as shown in Fig. 1 (b).

3) **Expansion:** After sufficient number of hits are found, each W-mer with exact match is expanded locally to both directions (left and right) by single letter per direction at a time as shown in Fig. 2. The expansion continues until the scoring no longer gets improved. The scoring technique of the algorithm is based on the Point Accepted Mutations (PAM) matrices, that are used to examine which amino-acid substitutions are biologically accepted [15]. However, due to the complexity of this technique, a simpler and biologically acceptable scoring method is generally used. In the expansion stage, each new match will results in addition of 5 points to the score and any mismatch will lead to reduction of 4 from the score as depicted in Fig.2. The expansion ceases when the score calculated after an expansion becomes less than the score before expansion. The sub-strings with maximum scores are selected and are called HSPs. The HSPs and the corresponding score are reported as the final output of the algorithm.

IV. Proposed Architechture

The overall architecture of HBlast library targeting on-premises implementation is as depicted in Fig. 3. It is implemented on an off-the-shelf vendor supported FPGA development board (Xilinx VC709). This board supports a PCIe x8 Gen.3 interface which theoretically supports up to 7.88 GB/s/direction bandwidth. It has 8GB external DRAM, supporting up to 933MHz clock frequency. The implementation consists of an IP core (Dyract) to interface with a host computer, a DRAM controller for interfacing with the external DDR memory and the module implementing the core BLAST

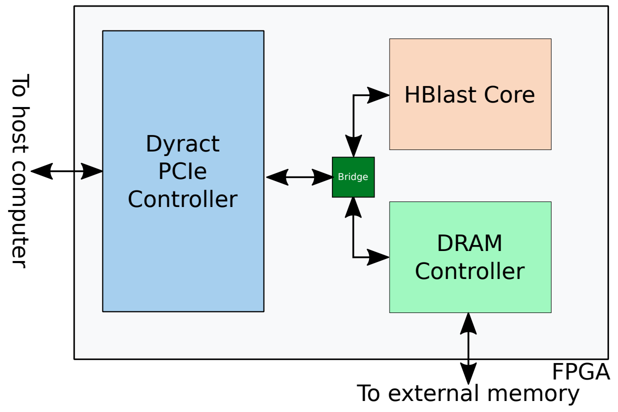


Fig. 3. Overall architecture of HBlast targeting on-premise FPGA implementation

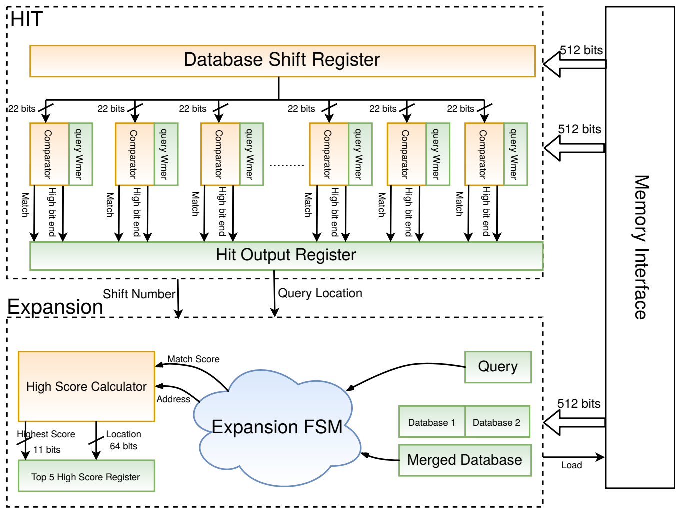


Fig. 4. Detailed architecture of the HBlast core module showing the Hit and Expansion submodule and their interconnection

algorithm. Details of each sub-module is discussed in the subsequent sections.

Our current implementation targets only DNA sequence searching which could be augmented with support for protein searching in the future. Since there are only 4 types of nucleobases encountered in DNA sequences (adenine (A), guanine (G) cytosine (C), and thymine (T)), only two bits are used to represent each base. This considerably reduces the memory requirement for storing the database compared to traditional character-based storage since each character takes one byte of data. The entire database is initially stored in the external DRAM memory of the FPGA-board and the query sequences are stored in the FPGA internal memory.

A. PCIe Controller

We use our previously developed open-source FPGA PCIe controller called Dyract to interface with the host computer [19]. The PCIe interface is used to send the database and query strings to the FPGA as well as to receive back the HSPs and the corresponding scores. This interface can provide up to 6.5 GB/s throughput, nearly saturating the available PCIe bandwidth. The PCIe controller is interfaced with both the memory controller as well as the BLast core. Depending upon the stream type, data is sent either to the external DRAM or the internal registers of BLast core through a bridge logic.

B. DRAM Controller

The DRAM controller internally uses Xilinx memory interface generator (MIG) IP core [20]. Present implementation instantiates a single MIG controller thus supporting up to 4GB external memory on VC709. The interface operates at 200MHz with 512-bit interface providing a maximum of 12.5 GB/s throughput. The throughput of the memory interface is very critical to the overall performance since the database sequences are continuously read from the external memory for comparing with the query sequence. Multiple levels of pipelining is implemented to support the required memory clock performance.

C. BHlast Core

This module implements the core BLAST algorithm discussed in Section III. For the current implementation, the maximum query size supported is 512bits (256 bases) with up to 4GB database. This can be extended to larger size by instantiating more number of basic compute units. The database is stored in the external DRAM memory and the query sequence is stored in the internal *queryReg* register. The two main sub-modules of the HBlast core are the *Hit* and *Expansion* modules as shown in Fig. 4. As their names state, these blocks are responsible for tasks like finding hit (match), expanding the sequence and linking blocks.

For the nucleotides, W-mer size is 11 letters [1], so by *Eq.1* there are 246 W-mers in 256-letter query. Each of them must be compared with each W-mer of the database. Comparison between one database and one query W-mer at a time is not effective in terms of latency. In HBlast architecture this operation is parallelized by introducing 246 comparators (one for each query W-mer). All comparators have common 22-bit (11-letter) database W-mer fetched from external memory. This input is controlled by the *Database Shift Register*, which performs a 2-bit shift operation after every clock cycle. The expansion module will be activated if output of any of these comparators is high, indicating a valid match has been found between the database and the query.

The expansion and hit are sequential operations, meaning the expansion will start only if the hit stops and other way around. A single expansion block is used to restrict the FPGA resource utilization and thus to improve the clock performance. Once a valid match is found, the hit module sends information regarding the location of the match in the query and the number of performed shift operations to the expand module.

D. Hit

The hit module compares W-mers from the database with W-mers in query in parallel to find exact matches. The module

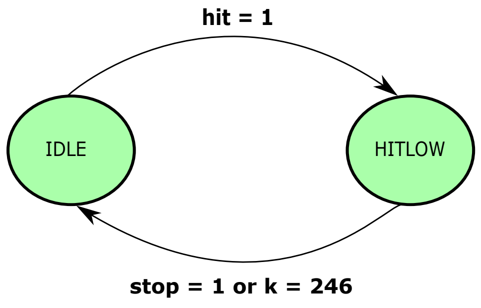


Fig. 5. FSM diagram for module Hit

implements a finite state machine (FSM) with two states as shown in Fig.5. In the Idle state, the module takes a 22-bit sequence from the database and compares each W-mer with the corresponding W-mers in the query sequence via 246 comparators. Due to the parallel architecture, the comparisons happen simultaneously which significantly improves the latency. If at least one exact match is detected, the signal *hit* is activated and the process passes to the next state, (*HitLow*). It stays in this state while expansion process is being performed. When expansion is over, the FSM goes back to the *Idle state* and verifies if there are other matches of the W-mers within the query. If there is a match again, the expansion process will be repeated, if there is no more match found, the loaded 512 bits of database is shifted by 2 bits and next 22 bits of sequence in database are compared with the query. When all W-mers in the database are compared, the next 512 bits of the database are loaded from the external memory and process is repeated.

E. Expansion

The Expansion Finite State Machine expands the exactly matched sequences of the database to both sides by comparing it with the query. The outputs from this module are the start and end locations of the most matching five expanded sequences and their high scores. The FSM consists of 6 states as shown in Fig. 6.

1) **Idle**: The FSM stays in this state until it receives the *start* signal signal from the *hit* module.
2) **Wait**: In this state module waits until it receives the database portion from the external DRAM.
3) **Load 1**: This state loads the 512-bits of database to a register. Then it passes to the next state according to the shift number by following logic: if the shift number is less than 199 or more than 290, it goes to state **Load 2**; otherwise it goes to the **Expand** state directly.
4) **Load 2**: This state takes another 512-bit piece of database located either before or after the database with exact matched sequence. This is to satisfy the boundary condition, where an exact match is detected at the boundary of the current database sequence stored in the FPGA. In order to expand, the next piece of database has to be fetched from the external memory. The starting address of the next sequence is calculated within the state and it goes to the next state which waits for data from the DRAM.

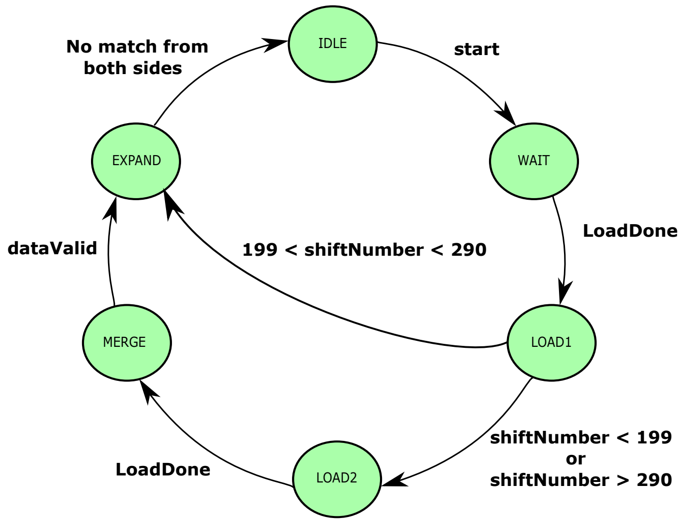


Fig. 6. FSM diagram for module Expansion

5) **Merge**: In this state the 512-bit sequence of the database is merged with the next 512-bit sequence to satisfy the boundary conditions.
6) **Expand**: The exactly matched W-mers of database and query expands to both sides by 2 bits each clock.

The initial score of HSP is 55 (obtained from 11*5). If next 2 bits are matched 5 is added to the HSP score, otherwise 4 points are deducted. Threshold value is chosen to be 200 bits, meaning that the sequence can be expanded to both sides by 200 bits each. Accordingly, maximum HSP can be 1055. The threshold value is set to 200 since the maximum query sequence size is 512-bits. The expansion is performed with all the matching W-mers of the query with the database. And as the final output the 5 sequences with the highest score, that is 5 most matching sequences of the query to database, is obtained with their positions as start and end locations in the database. However, the operation stops when the exact matching sequence with highest score of 1055 is detected. There is no need of further matches since the that is the maximum possible score for an HSP.

F. Memory Interface

The module Memory Interface is a control unit that interacts with modules Expansion FSM, Hit and bridge. It is an FSM of 7 states. The main functions of the module are following:

- The module takes addresses from the expansion or hit modules and sends the control signals to the DRAM controller.
- When it gets signal which indicates the absence or expiration in matches between query and 22-bit W-mer of the database from Hit module, it sends signal to shift the current piece of the database to the module.
- The module arbitrates the memory access between the Expansion and Hit modules.

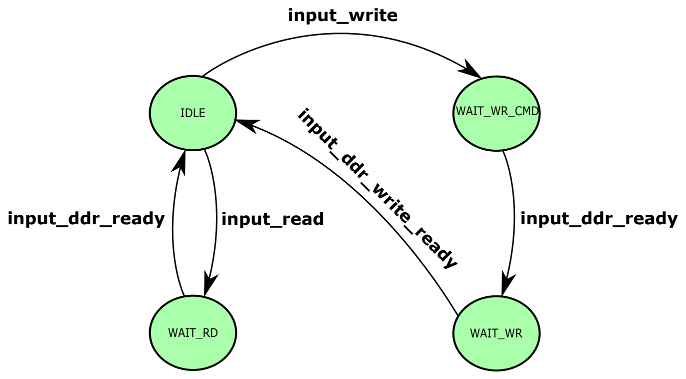


Fig. 7. FSM diagram for bridge

G. Bridge

The Bridge module implements the data width matching between the PCIe interface and the Memory Interface and arbitrates the memory access between the PCIe core and the BLast core. It consists of four states, refer to Fig. 7:

1) **Idle**. Depending on whether input signal is write or read, the state changes to Wait_wr_cmd and Wait_rd respectively. Write signal comes from PCIe when it wants to write query data and database into DRAM. While, read signal comes from two main modules (Hit and Expand) of the architecture when it wants to take data from DDR.
2) **Wait_wr_cmd**. This state waits for the acknowledgement from DDR, as it is received, input data is sent to DDR. Then, state changes to the next Wait_wr state.
3) **Wait_wr**. The FSM state waits for write_ready acknowledgment, and then increments the write address by 8. The DRAM interface from FPGA is 64-bit wide hence to write 512-bit sequence, 8 write operations are required.
4) **Wait_rd**. After receiving read ready signal from DRAM controller, it reads data 8 times and each time increments address by 8 for the same reason as explained above.

H. HBlast Cloud Architecture

Fig. 8 depicts the architecture of the HBlast module when implemented on an AWS-EC2 F1 FPGA instance [8]. The F1 instance is a Xilinx Ultrascale+ FPGA and supports PCIe Gen3 x16 interface to the host computer. Thus, it supports twice the bandwidth between the host computer and FPGA compared to VC709 implementation. Each instance has 64GB external DRAM (DDR4) memory support also. Amazon provides a *shell* infrastructure for the FPGA, which manages the communication between the FPGA and the host machine as well as between the FPGA and the external DRAM memory. This eliminates the need for the Dyract IP and the DRAM controller when targeting the design for the cloud infrastructure.

The bridge logic and the HBLast core are implemented in the *custom logic (CL)* portion for the F1 instance. The bridge logic is interfaced with PCIe master and slave interfaces of the shell as well as one of the master ports of the shell

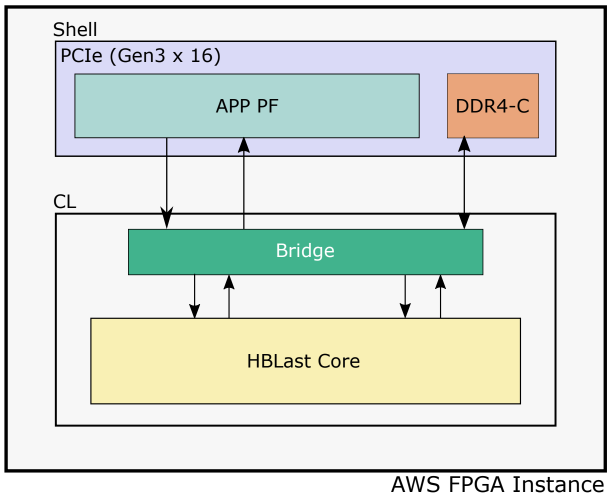


Fig. 8. HBlast architecture when targeting Amazon EC2 F1 FPGA instance

TABLE I
TABLE TITLE

FPGA module	Slice LUTs	Slice Regs	Block RAM Tile
HBlast	11430	10738	1
bridge	68	641	0
memoryInt	5321	4809	0
Expand	2342	2410	0
hitMem	2958	808	0
queryB	39	612	0
u_mig_7series_0	6002	4676	1

DRAM interface. Similar to the bridge logic of the on-premise implementation, this logic routes the data coming from the host server either to the external DRAM or to the internal memory of the HBLast core depending on the data stream type (database or query). The HBLast module instantiates the same modules as discussed before and runs at 200MHz clock frequency provided by the shell logic.

V. DISCUSSION AND ANALYSIS

In this section we discuss the implementation details of the HBlast architecture on FPGA. Detailed reconfiguration performance numbers are reported.

A. Implementation

Xilinx ISE was used for implementation. The proposed platform was implemented and hardware validated on a Xilinx VC709 evaluation board containing a Virtex-7 XC7VX690T-2FFG1761C FPGA. The static and reconfigurable regions on the FPGA are shown in *Fig.***??**.

B. Resource utilization

Resource utilization is shown in *Table I*. On the Virtex-7 FPGA the platform consumes about % of logic and BRAM utilization is about %.

TABLE II
RESOURCE UTILIZATION PER MODULE

Name	LUTs	FF	BRAM
bridge	66	642	0
memoryInt	3165	4821	0
Expand	2980	2451	0
highScore	381	697	0
hitMem	3134	1320	0
comparator	10	1	0
dbShiftReg	531	532	0
queryB	196	612	0

TABLE III
RESOURCE UTILIZATION OF THE ARCHITECTURE

Resource	Estimation	Available	Utiliztion %
LUT	5428	433200	1.25
FF	6062	866400	0.70
IO	104	850	12.24

C. Timing analysis

In VC709 board system clock is 200MHz generated by frequency oscillator. Therefore, there is no slack the maximum frequency will be same as the oscillators frequency. *Table IV* shows the timing report of our design.

The latency is a duration of taken time, which starts right after the reading query and ends when the search is complete. When the device starts reading the register, where query is stored, it waits until the signal that indicates the validity of query comes. Once the *queryValid* flag is high, the process of searching for the hits/matches is initiated, and it lasts until the end of the algorithm. In the architecture, the expansion process takes much more time than the finding the hit. In expansion process, if the exact match is found in the middle of the of 512-bits, that is there are at least 200 bits before or after the location of the match, the former or latter 512-bits of the database in DDR is not retrieved. Otherwise, if that condition is not hold, DDR will be accessed one more time to retrieve the additional 512-bits to allow the range of expansion be sufficient for search operation. Indeed, access time to DDR is relatively long process.

The proposed architecture was examined under two scenarios to identify the dependence of latency on the characteristics of the search in database. The first scenario was to determine the relationship between the latency and the location of the exact match in database. As it is depicted in Fig.9, x-axis represents the location of exact match in database, and y-axis shows the corresponding time to complete the search. The

TABLE IV
TIME AND FREQUENCY CHARACTERISTICS

Total slack, ns	Required time, ns	Arrival time, ns	Maximum frequency, MHz
-0.76	5.000	5.76	173.6

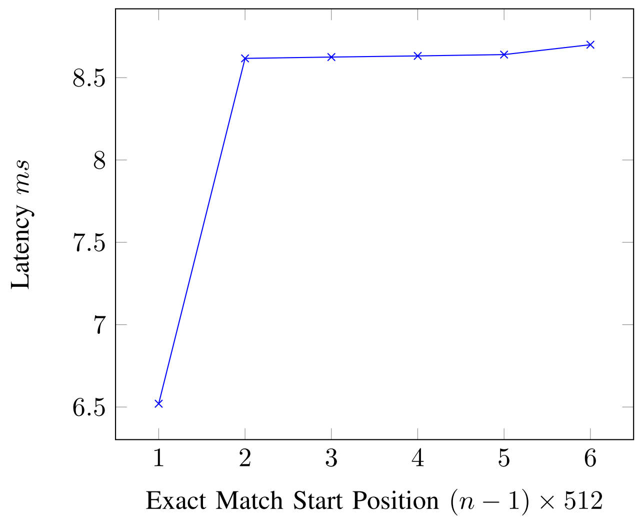


Fig. 9. Relationship between latency and position of exact match of sequence in database

matches are located in different 512-bits in database, thus, the matches are found in only a certain read operation from the database. The first point in the plot, refer to Fig.9, indicates the latency when the match is located in the first 512-bits of the database. The rest of the points, similarly, represents that the matches are located in second, third, forth, fifth and sixth 512-bits of the database respectively. It can be clearly seen that the latency for the first case (match in first 512-bits of database) differs from others. The latency in the first case is approximately 6.5 msec, and in the rest of cases it is roughly 8.6 msec. The difference in latency can be explained by the fact that the match is located in the first half of the 512-bits of the database. Thus, there is no information available before these 512-bits, as a consequence DDR is accessed only once and the expansion takes place only in that particular sequence. In the rest of the cases, there is additional information to retrieve from the DDR, so there are 2 accesses made to DDR. Hence, the latency is longer that the first case.

The second scenario was to test the architecture by increasing the number of matches between query and database. It can be seen in Fig.10 that the latency for the smallest number of matches is the lowest, and as the number of matches increases the latency becomes larger. Due to the fact that each subsequent case of increasing the number of matched W-mers is conducted by adding the short sequence of match, the difference between the latency does not increase dramatically. However, there is a clear tendency of latency to increase with the number of matches. The reason of the difference of the first case form the rest is the same as it was discussed in previous case scenario: the expansion process does not access DDR twice, because there is no available information before the first 512-bits of database.

VI. CONCLUSION AND FUTURE WORK

In this paper, we propose the FPGA-based implementation of BLAST algorithm called HBlast. The proposed architecture

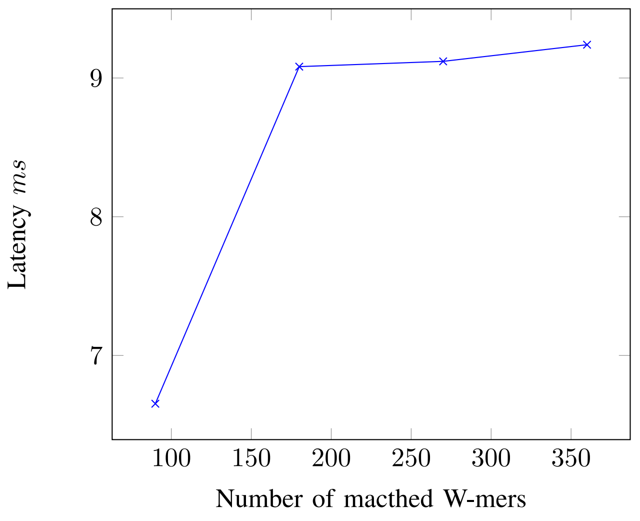


Fig. 10. Relationship between latency and number of exact match of sequence in database

utilizes the parallel search processing, the efficiency of which is determined in the analysis part.

Indeed, this project aims to be one of the several FPGA-based implementations of search algorithms. For the time being, the proposed architecture is available as open source and can be accessed by following source [**?**]. The larger project will constitute a library of hardware accelerators targeting cloud-based FPGA instances. The accessibility of the accelerators in the public domain will enable interested organizations to achieve hardware acceleration through cloud-based infrastructures such as Amazon AWS FPGA instances. In fact, this will be effective and suitable for research centres with limited computing facilities. Moreover, by having a communication infrastructure for interfacing the accelerators with standard PCIe and DRAM interfaces, accelerators, such as the proposed one in this paper, could be also used in personal FPGAs.

References

[1] S. Kasap, K. Benkrid, and Y. Liu, "Design and Implementation of an FPGA-based Core for Gapped BLAST Sequence Alignment with the Two-Hit Method." *Engineering Letters*, vol. 16, no. 3, 2008.

[2] X. Guo, H. Wang, and V. Devabhaktuni, "A systolic array-based FPGA parallel architecture for the BLAST algorithm," *ISRN bioinformatics*, vol. 2012, 2012.

[3] S. Datta, P. Beeraka, and R. Sass, "RC-BLASTn: Implementation and evaluation of the BLASTn scan function," in *Field Programmable Custom Computing Machines, 2009. FCCM'09. 17th IEEE Symposium on*. IEEE, 2009, pp. 88–95.

[4] M. Yoshimi, C. Wu, and T. Yoshinaga, "Accelerating BLAST Computation on an FPGA-enhanced PC Cluster," *IEEE*, 2016.

[5] H. Abelsson, G. Sandberg, and S. Mohl, "Accelerating NCBI BLAST," *Cray User Group*, 2007.

[6] F. B. C. H. Mohd, "Design and Implementation of FPGA-based DNA Sequence Alignmnet Accelerator," Master's thesis, Universiti Tun Hussein Onn Malaysia, 2013.

[7] L. Wienbrandt, S. Baumgart, J. Bissel, F. Schatz, and M. Schimmler, "Massively parallel FPGA-based implementation of BLASTp with the two-hit method," *ELSEVIER*, 2011.

[8] Amazon. (2018) AWS EC2 FPGA Hardware and Software Development Kit. [Online]. Available: https://github.com/aws/aws-fpga

[9] NCBI. Basic Local Alignment Search Tool. [Online]. Available: https://blast.ncbi.nlm.nih.gov/Blast.cgi

[10] L. Wienbrandt, S. Baumgart, J. Bissel, F. Schatz, and M. Schimmler, "Massively parallel FPGA-based implementation of BLASTp with the two-hit method," *Procedia Computer Science*, vol. 4, pp. 1967–1976, 2011.

[11] T. Oliver, B. Schmidt, and D. Maskell, "Hyper customized processors for bio-sequence database scanning on FPGAs," in *Proceedings of the ACM/SIGDA international symposium on Field-programmable gate arrays*. ACM, 2005, pp. 229–237.

[12] X. Guo, "Design of a systolic array-based FPGA parallel architecture for the BLAST algorithm and its implementation," Master's thesis, The University of Toledo, 2012.

[13] J. D. Buhler, J. M. Lancaster, A. C. Jacob, R. D. Chamberlain *et al.*, "Mercury BLASTN: Faster DNA sequence comparison using a streaming hardware architecture," *Proc. of Reconfigurable Systems Summer Institute*, 2007.

[14] D. Lavenier, G. Georges, and X. Liu, "A reconfigurable index FLASH memory tailored to seed-based genomic sequence comparison algorithms," *The Journal of VLSI Signal Processing Systems for Signal, Image, and Video Technology*, vol. 48, no. 3, pp. 255–269, 2007.

[15] E. Sotiriades and A. Dollas, "Design space exploration for the BLAST algorithm implementation," in *Field-Programmable Custom Computing Machines, 2007. FCCM 2007. 15th Annual IEEE Symposium on*. IEEE, 2007, pp. 323–326.

[16] M. C. Herbordt, J. Model, Y. Gu, B. Sukhwani, and T. VanCourt, "Single pass, BLAST-like, approximate string matching on FPGAs," in *Field-Programmable Custom Computing Machines, 2006. FCCM'06. 14th Annual IEEE Symposium on*. IEEE, 2006, pp. 217–226.

[17] B. Harris, A. C. Jacob, J. M. Lancaster, J. Buhler, and R. D. Chamberlain, "A banded Smith-Waterman FPGA accelerator for Mercury BLASTP," in *Field Programmable Logic and Applications, 2007. FPL 2007. International Conference on*. IEEE, 2007, pp. 765–769.

[18] S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman, "Basic local alignment search tool," *Journal of Molecular Biology*, vol. 215, no. 3, pp. 403 – 410, 1990. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0022283605803602

[19] K. Vipin and S. A. Fahmy, "DyRACT: A partial reconfiguration enabled accelerator and test platform," in *2014 24th International Conference on Field Programmable Logic and Applications (FPL)*, Sep. 2014, pp. 1–7.

[20] *7 Series FPGAs Memory Interface Solutions*, Xilinx Inc, Mar. 2011.