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*—A numerous FPGA-based designs were presented for accelerating similarity search algorithms in Bioinformatics. Those design were built according to many algorithms, however, the BLAST algorithm is the most widely used algorithm transferred to reconfigurable hardware. This paper presents the design and implementation of the FPGA-based core for BLAST algorithm with parallel search for match between sequences. This results in a high performance and effective FPGA implementation, which outperforms the equivalent software implementation. The new architecture and detailed performance results are presented in this paper.**

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

I. Introduction

Biological sequence alignment search 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 of 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 the running the software tools and will be beyond the capabilities of small hospitals and research centres. 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 time multiplex portions of the same algorithm to fit in a smaller device. The reprogrammability also make them attractive compared to other more expensive hardware implementation such as ASIC [5].

The efficiency of the search algorithm is essential, and probably is even more important than 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 as a consequence, even algorithms like BLAST cannot remain efficient to meet 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 a part of a larger project, which develops a library of hardware accelerators targeting cloud-based FPGA instances. The accelerators will be provided in the public domain, which enables hospitals and research centres with limited computing facilities to achieve hardware acceleration by hosting these accelerators on the cloud infrastructure such as Amazon AWS FPGA instances []. 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 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 Xilinx Virtex-7 FPGA
- Characterization of the proposed platform in terms of FPGA resource consumption and performance

The rest of this paper is organized as, Section 2 discussed the related work, Section 3 discusses the 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. Background and Related Work

FPGAs are applied in sequence comparison in Large Sequence Databanks for the reason that they significantly improve performance due to parallel architecture. There are number of proposed and applied implementations of BLAST algorithm on FPGA in order to address the issue of biological sequence alignment [8]. The following examples are the proposed hardware implementations of BLAST algorithm: Mercury BLASTn [9], Mercury BLASTp [10], Tree-BLAST [11], RC-BLAST [3], FPGA/FLASH Accelerator [12] and Multiengine BLASTn Accelerator [13].

The architectures mentioned above are based on the Word Position Record-Based Search (WPRBS) method [2], which stores words of query sequence in a constructed storage table and compares the subject sequence with storage table to detect the match. The strategy is widely applied, however, performance and searching capacity limitations are still extant. The reason is that during one clock cycle only one word can be searched, and thus more than one hit per clock cycle cannot be detected. 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 [9] 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. Apparently, performance of the system will be low: accessing external memory SRAM for getting subject sequence from database every time consumes significant amount of time.

Performance could be improved by FPGA/FLASH [12], which minimizes accesses to the external memory by enabling to detect several exact matches/hits simultaneously. It is due to index structure of the database: words in the sequence associated with their position in the database and neighboring elements. However, huge space in the memory 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 order to solve the timing isuse, the architecture Multiengines BLASTn [13] is constructed in such way that compares 64 subject sequences simultaneously due to its 64 identical computing units. The architectures described above are based on WPRBS.

There is another method based on systolic array that does not construct tables. Needleman-Wunsch algorithm [14] and dynamic programming algorithm that apply systolic array was implemented on SPLASH2 by D. Hoang et. al. [15]. Another algorithm that uses systolic array called Smith Waterman matching algorithm was implemented by S. Guccione et. al., also at Virginia Tech and Nanyang Techological University [16].

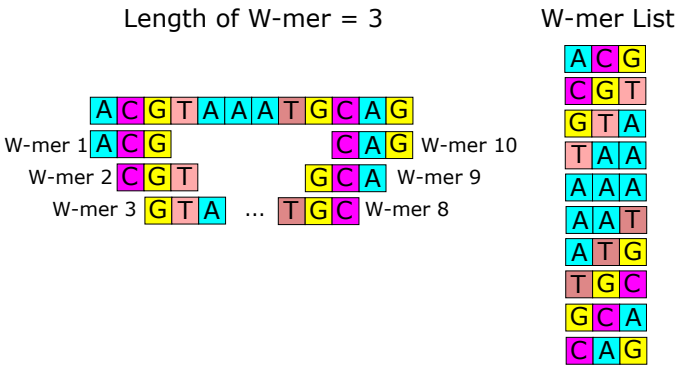

Fig. 1. Example of BLAST Algorithm's First Step: Pre-Processing of the Query.

III. Blast Algorithm

BLAST algorithm is used to search for similar parts of sequence between database and query. There are several implementations of BLAST based on their processing data: nucleotides having 4 letter alphabet and amino acids having 20 letters. The inputs to BLAST are two sequence - database, which consists of a huge amount of data, and a query, which is to be compared with database and find the similar parts of sequence. The output of algorithm are the degree(score) of similarity of the aligned parts and their corresponding location in the sequences. Each of matched pair in database and 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 the hit*. Obtained small words of query are compared to the data in database to determine the exact match.
- *Expansion of comparison around hits*. In the location where the sequences coincide, the comparison is continued from both sides and HSP is calculated.

Step 1. In this step, query is divided into a list of substrings, refer to *Fig.1*. These substrings are called W-mers of length W [13]. Let us suppose we have a query DNA sequence ACGTAAARGCAG of length 12 and W-mers of length 3 [13]. Since the w-mers are contiguous substrings, there are in total 10 W-mers. Indeed, the number of W-mers are calculated as

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

The substring ACG will be the first w-mer and CGT is the second and so on.

Step 2. 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. Every found exact match is recorded as a ”hit” and saved to proceed further to step 3, refer to *Fig.2*.

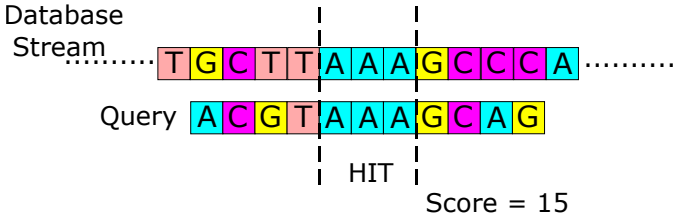

Fig. 2. Example of BLAST Algorithm’s Second Step: Search for Exact Match/Hit.

Step 3. After all hits are found, each W-mer/substring is expanded locally to both directions by single letter per direction at a time, refer to *Fig.3*. The expansion holds until the scoring no longer gets improved.

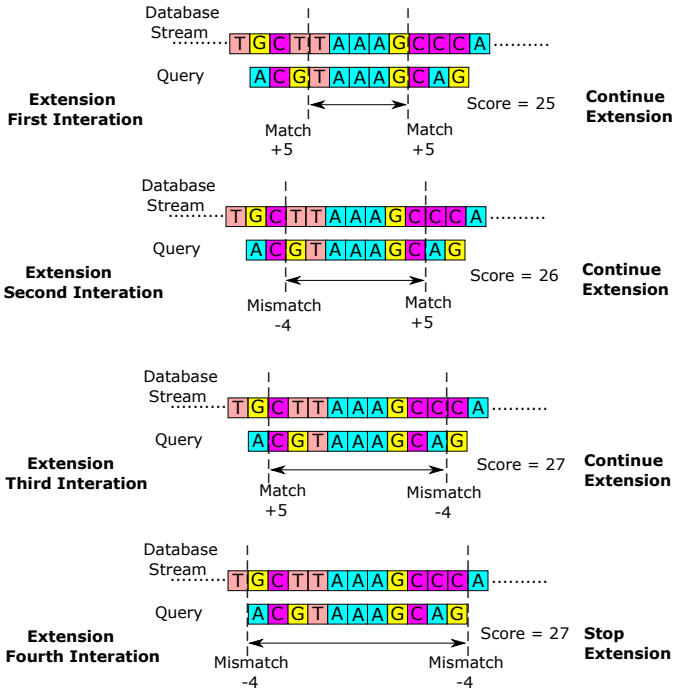

Fig. 3. Example of BLAST Algorithm’s Third Step: Expansion of Comparison around Hits

The scoring technique of the algorithm is actually based in the Point Accepted Mutations(PAM) matrices, that are used to examine which amino-acid ”substitutions” are biologically accepted [13]. However, due to the complexity of this technique, the simpler and biologically acceptable (correct in biological point of view) scoring method will be used. In the expansion stage, each new match will results in addition of 5 and any mismatch will lead to reduction of 4, refer to Fig.3.

IV. Proposed Architechture

The HBlast architecture, proposed in this paper, was designed for small query (512-bit = 256 letter) but for large sized data base (4GB). First, database sequence will be saved in double data rate(DDR) SDRAM and query sequence will be saved in register *queryReg*. Then the actual alignment operation will start.

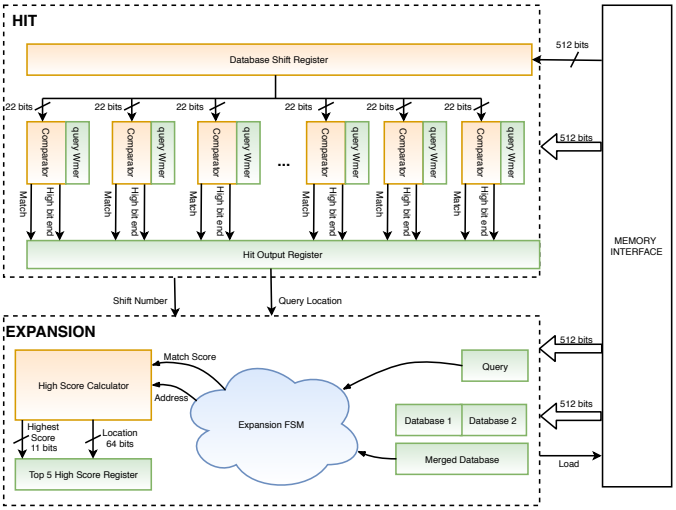

Fig. 4. Proposed Architecture / HBlast machine.

The two main blocks of the HBlast architecture are *Hit* and *Expand*, while *Memory Interface* acts as a control logic for them, refer to *Fig.4*. As their names state, these blocks are responsible for tasks like finding hit (match), expanding the sequence and linking blocks.

For the nucleotide W-mer size is 11 letters, so by *Eq.1* there are 246 W-mers in 256-letter query. Each of them must be compared with one W-mer of the database. Comparison of one database and one query W-mer at a time is not effective in terms of timing. In HBlast architecture this operation is parallelized by introducing 246 comparators (one for each query W-mer). All comparators have one common input: 22-bit (11-letter) database W-mer. This input is controlled by *Database Shift Register*, which perform 2-bit shift after each iteration. The expansion will start if output of any of these comparators is high.

The expansion and hit are sequential operations not parallel, meaning that the expansion will start only if the hit stops and other way around. After the hit operation finishes, it will send information about the location of the match in query and number of performed shift operations. Loading the data from memory interface is controlled by signal *load*. The number of state changes of the signal (low or high) is then decided by examining *shiftNumber*. There can be at most two read operations for one iteration. Each time the read data will be saved in 512-bit *Database* register and then merged into 1024-bit register, *MergedDatabase*. Then the expansion will work by the logic described in the following *Expand Finite State Machine* section. At the same time the block *High Score Calculator* will calculate the match score and keep track of the five HSP (with their location in database and query).

A. Hit

Hit (hit.v module) is one of the two main modules in proposed architecture, which is aimed to compare W-mers from the database with W-mers in query in parallel and to find exact match. The module is a finite state machine (FSM) of two states, refer to *Fig.5*. In the first state (*Idle state*), the module takes 22 bits of sequence from the database and compares the W-mer with each W-mer from the query sequence via 246 comparators. Parallel architecture of FPGA allows performing the task simultaneously, which significantly improves timing. Then, if at least one exact match is detected, the signal *hit* is received 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 process again goes to *Idle state* and verifies if there are other matches of the W-mer 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 gone through all the processes described above. When all W-mers in the database are compared, then next 512 bits of the database are loaded and process is initiated again.

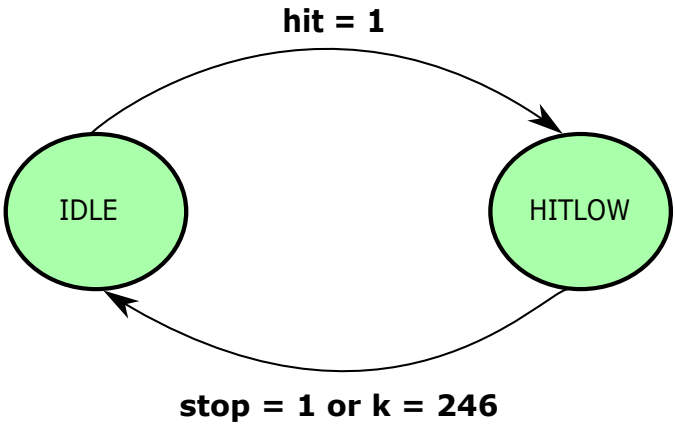

Fig. 5. FSM diagram for module Hit

B. Expansion Finite State Machine

Expansion Finite State Machine (ExpansionFSM.v module) is a second main module that expands exactly matched sequence of the database to both sides by comparing it with the query. The outputs of the module are start and end locations of the expanded matching sequence and its high score. Expand FSM is a FSM of 6 states, refer to *Fig.6*:

1) **Idle**: It stays in in this state until the signal that indicates the start of the expansion comes.
2) **Wait**: In this state module waits until the signal of loading data from DDR is received.
3) **Load 1**: This state loads the 512-bit piece of database to 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 state **Expand** directly. .

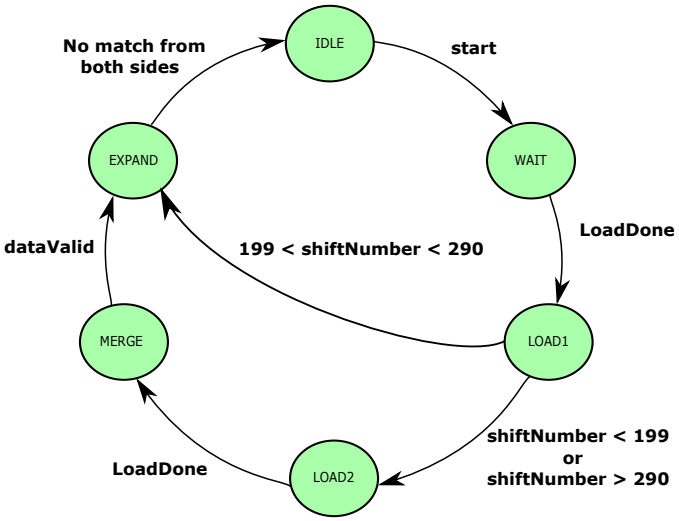

Fig. 6. FSM diagram for module Expansion

4) **Load 2**: This state decides to take another 512-bit piece of database located either before or after the database with exact matched sequence. The address of the piece is calculated within the state and it goes to the next state which waits loading of the piece.
5) **Merge**: In this state the 512-bit piece of database merges with another 512-bit sequence. This is done in order to broaden expansion's scopes.
6) **Expand**: The exactly matched W-mers of database and query expands to both sides by 2 bits each clock.

The initial value of HSP is 55 (obtained from 11*5). If next 2 bits are matched 5 is added to the HSP, otherwise 4 points are subtracted. 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.

C. Memory Interface

The module Memory Interface (memInt.v module) 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 loads respective data.
- Module receives and sends control signals. For instance, when it gets signal which indicates 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 allows to perform the whole process sequentially and to avoid confusion of dataflow between ExpandFSM and Hit modules. For example, the address of the data that must be passed from Hit to ExpandFSM are calculated in the module and are used to load necessary data.

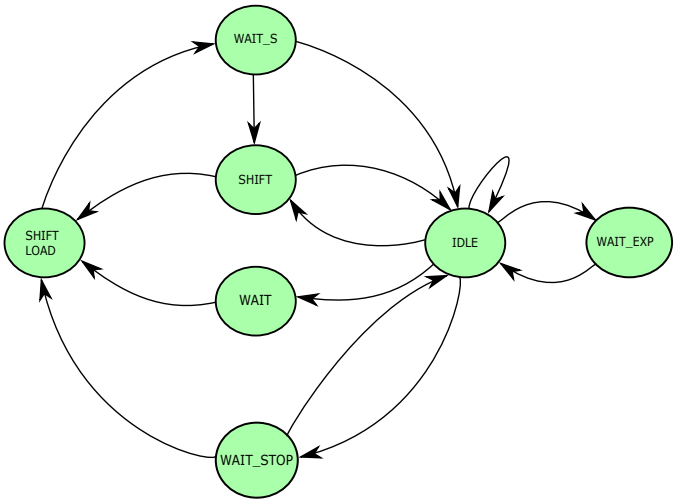

Fig. 7. FSM diagram for Memory Interface

D. Bridge

The Bridge module is the module which makes data width matching between FPGA board with 7 series Memory Interface. It consists of four states, refer to Fig. xxx:

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 DDR SDRAM. 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 present state waits for write_ready acknowledgment, then increments the write address, which is byte addressable, by 8. The reason for this is that each time we write 64 bits of data, which is 8 bytes. For instance, in order to write 512-bit sequence, 8 writes are needed. After all these operations, it goes to Idle state.
4) **Wait_rd**. After receiving acknowledgment from DDR that DDR is ready, it reads data for 8 times and each time increments initial address by 8 for the same reason as explained above. After these operation are completed, the state is changed to Idle.

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.***??**.

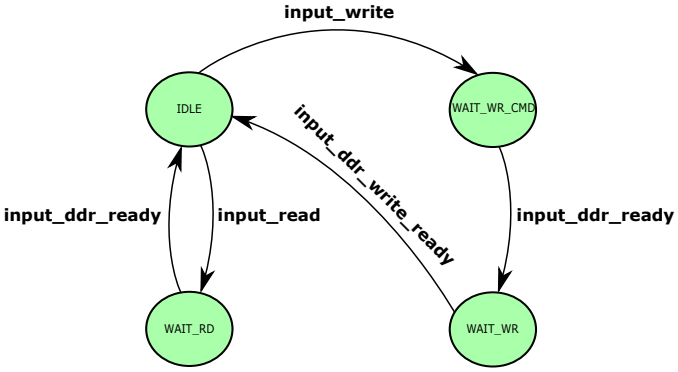

Fig. 8. FSM diagram for bridge

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

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

VI. Conclusion and Future Work

- query length is limited by 512 bits - expansion waits for hit, hit waits for expansions expansions are not parallel

In paper [**?**]

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.

TABLE II
Table Title

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

Total slack, ns	Required time, ns	Arrival time, ns	Maximum frequency, MHz
1.963	5.000	-3.037	143.6

[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] 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.

[9] 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.

[10] 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.

[11] 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.

[12] 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.

[13] 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.

[14] G. H. Gonnet, M. A. Cohen, and S. A. Benner, "Exhaustive matching of the entire protein sequence database," *Science*, vol. 256, no. 5062, pp. 1443–1445, 1992.

[15] D. T. Hoang, "Searching genetic databases on splash 2," in *FPGAs for Custom Computing Machines, 1993. Proceedings. IEEE Workshop on*. IEEE, 1993, pp. 185–191.

[16] C. W. Yu, K. Kwong, K.-H. Lee, and P. H. W. Leong, "A smith-waterman systolic cell," in *International Conference on Field Programmable Logic and Applications*. Springer, 2003, pp. 375–384.