Local Alignment Search in Genetic Sequences on a Low-Cost FPGA

Timm Bostelmann, Thomas Fabian Starke and Sergei Sawitzki

FH Wedel (University of Applied Sciences)
Wedel, Germany
Email: bos@fh-wedel.de, starke.thomas@yahoo.de, saw@fh-wedel.de

***Abstract*—The search for local alignments in genetic sequences is a common challenge in the field of bioinformatics. The problem is to find similar subsequences in genetic sequences of different lengths. Usually, the search is done in a genome database that contains hundreds of millions of sequences and rising. Due to the large amount of data, the speed is of a high concern. The search for a local alignment between a query-sequence and a database-sequence is usually done with the Basic Local Alignment Search Tool (BLAST) algorithm. In this work, an implementation of an accelerator for the BLAST algorithm on a low-cost Field-Programmable Gate Array (FPGA) is presented. The data is processed in a tree-like hardware architecture. The advantages and disadvantages of the presented approach are shown and discussed. Finally, an outlook is given on the pending issues of the current implementation. The main contribution of this paper is the focus on an implementation with support of low-cost hardware.**

Keywords–FPGA; BLAST; DNA; local alignment

I. Introduction

In the field of bioinformatics, the comparison of genetic sequences is as challenging as it is important. Usually, a query-sequence is compared with a set of sequences that are stored in a genome-database. The goal is to find partial or complete similarities. This is for example useful if an unknown virus is analyzed because properties of the unknown virus can be derived from similarities to known entities.

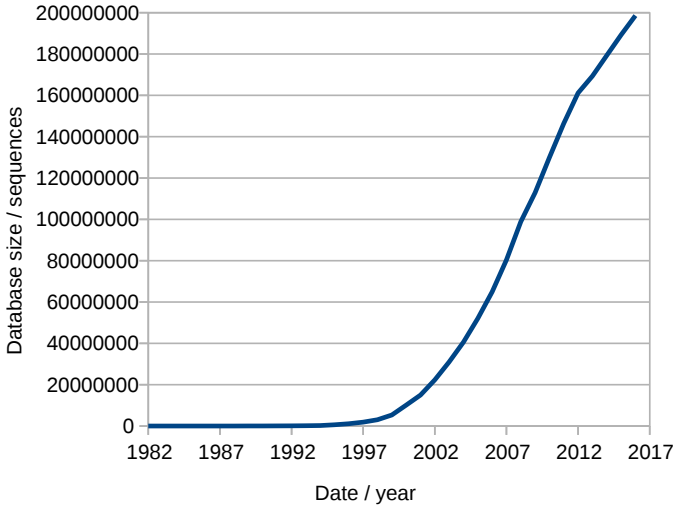

Figure 1. Chart of the number of genetic sequences stored in the genome-database of the National Center for Biotechnology Information (NCBI) over time [1].

Over the last decades, the size and number of genetic sequences stored in genome-databases has risen considerably, as shown in Figure 1. Further growth of the databases is to be expected, due to the shrinking costs of genome sequencing (see Figure 2). As a result, a fast and efficient implementation of the search-algorithms is mandatory.

The preferred method for the comparison of genetic sequences of different lengths is the local alignment search. Where the global alignment search is well suited to find similarities in sequences of equal or similar length, the local alignment search is utilized to find similar subsequences in genetic sequences of different lengths. However, this process conveys an extremely high computational effort. An established approach is the BLAST algorithm which has been introduced by Altschul et al. [3].

There have been several works on the acceleration of the local alignment search with high-performance FPGAs [4]–[6]. Usually, the proposed accelerators utilize clusters of several high-performance FPGAs. There are even industrial BLAST-accelerators based on such hardware available, like [7]. However, in this work an implementation on a low-cost FPGA development and education board is presented and analyzed. It is based on a tree-like data-flow [8] architecture.

The rest of this work is structured as follows. In Section II, the principles of the BLAST algorithm are introduced. Based on this introduction, in Section III, an implementation of the BLAST algorithm on a low-cost FPGA board is described. In Section IV, the results of this implementation are presented. Furthermore, the advantages and disadvantages compared to an implementation in software are discussed. Finally in Section V, this work is summarized and a prospect to further optimization is given.

II. Background

The BLAST algorithm is used for the search of similarities (i.e., local alignments) between a query-sequence and the sequences of a genome-database. It generates a list of positions or regions that are similar between the query-sequence and the database-sequence. Additionally, the significance of every hit is generated as a measure of the similarity.

The BLAST algorithm can be divided into five steps which are described in the following. It has to be remarked that the first four steps have to be executed for all entries in the genome-database and are therefore especially time-sensitive.

Step 1: Creation of the hit-matrix

For the search of a query-sequence w in a database-sequence u with a scoring-function $\sigma(\alpha \rightarrow \beta)$ both sequences are segmented into overlapping words of the length $q \in \mathbb{N}$.

 ISBN: 978-1-61208-585-2

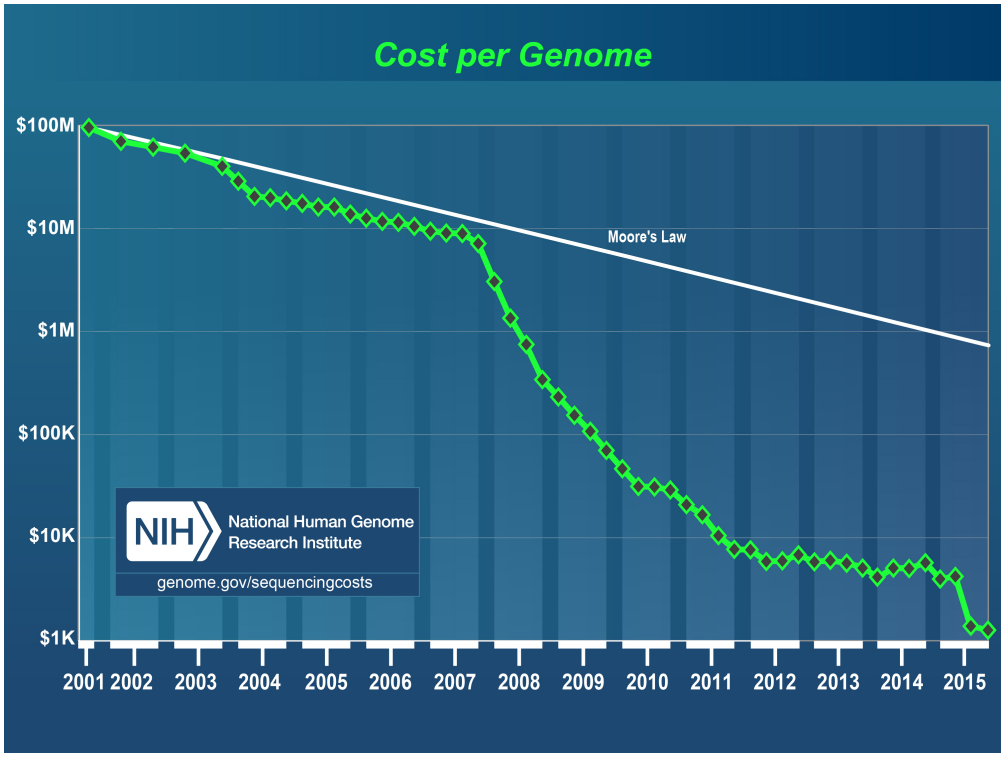

Figure 2. Chart of the cost per sequenced genome over time [2].

The following example is assuming a word length of $q = 3$:

$$\text{AVKTCSGA} \Rightarrow \{\text{AVK, VKT, KTC, TCS, CSG, SGA}\} \quad (1)$$

The resulting set of words is called w-mers. The generated words are called q-words. The scoring-function σ is used to determine the degree of similarity between a symbol of the query-sequence and a symbol of the database-sequence. Depending on the sequence-type usually either the Block Substitution Matrix (BLOSUM) or the Point Accepted Mutation (PAM) matrix is used to determine the degree of similarity. Deletions and insertions are handled as follows:

$$\sigma(\text{“-”} \rightarrow \beta) = \sigma(\alpha \rightarrow \text{“-”}) = -\infty \quad (2)$$

Assuming i and j are the indexes in the query- and the database-sequence, a pair (i, j) is a hit, if for a threshold k the following inequation is true:

$$i \in \{0 \ldots |u| - q + 1\} \quad (3)$$

$$j \in \{0 \ldots |w| - q + 1\} \quad (4)$$

$$score_\sigma(\underbrace{u[i \ldots i+q-1]}_{\text{q-word in u}}, \underbrace{w[j \ldots j+q-1]}_{\text{q-word in w}}) \geq k \quad (5)$$

Every q-word of the query-sequence is compared with every q-word of the database-sequence by the scoring-function σ. The results are stored in a hit-matrix, as shown in Figure 3.

Step 2: Extraction of relevant hits

To optimize the following steps three and four, the relevant hits in the hit-matrix are identified. Therefore, all hits that are located adjacently on a common diagonal are grouped like shown in Figure 4. Usually, a hit-length d is specified as the minimal length of the diagonals. The gray hits in Figure 4 are not grouped and will therefore be sorted out.

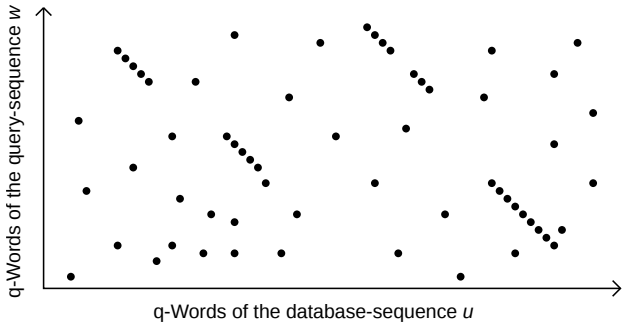

Figure 3. An exemplary hit-matrix generated by the first step of the BLAST algorithm.

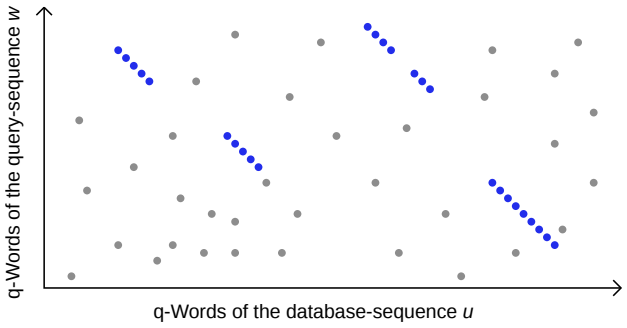

Figure 4. An exemplary hit-matrix after the extraction of relevant hits by the second step of the BLAST algorithm.

 ISBN: 978-1-61208-585-2

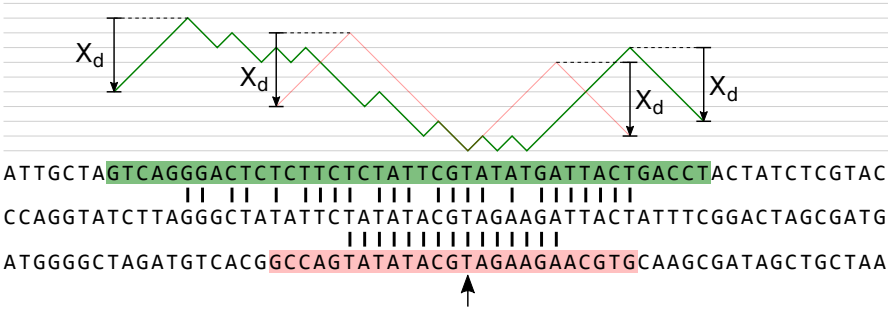

Figure 5. An example of a gapped extension for two different sequence pairs.

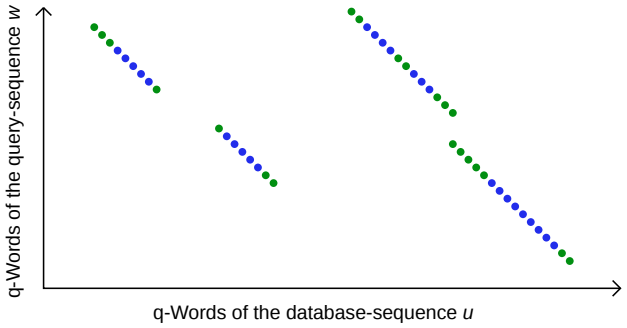

Figure 6. An exemplary hit-matrix after the ungapped extension by the third step of the BLAST algorithm.

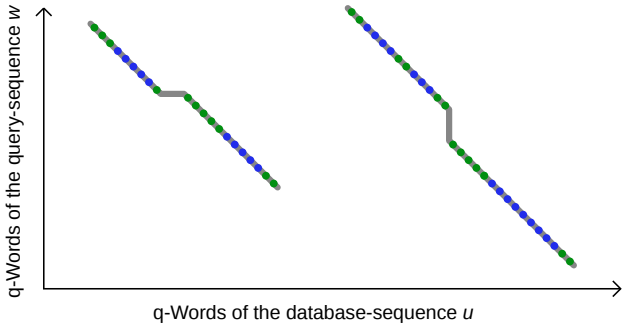

Figure 7. An exemplary hit-matrix after the gapped extension by the fourth step of the BLAST algorithm.

Step 3: Ungapped extension

If a pair of hits is aligned on the same diagonal and its distance is lower then the maximum distance δ, a point (i, j) between the hits is extended in both directions on the common diagonal. The sensitivity of the extension is determined by the drop-off parameter $X_d \geq 0$. For a leftward extension, the sequences $u[1 \ldots i-1]$ and $w[1 \ldots j-1]$ are compared. For a rightward extension, the sequences $u[i \ldots |u|]$ and $w[j \ldots |w|]$ are compared. The scores of those comparisons are summed up and the maximum X_{max} is stored. If this value is lower than the drop-off parameter $(X_{max} - X_d)$, the extension is stopped. The sequences generated by this stage are called maximum-scoring segment pair. An exemplary result is shown in Figure 6.

Step 4: Gapped extension

In this step, regions of high similarity with acceptable gaps between two sequences are determined. This is for example useful to skip insertions or deletions in a sequence, which can be for example caused by mutations. In the hit-matrix, the result is a shift of the diagonal (see Figure 7).

After identifying a region of high similarity, a point (i, j) between two hits is chosen as a starting point. Then, this point is expanded towards the two selected hits. The symbols of the two sequences are compared and the result is summed up, where a match corresponds to +1 and a mismatch corresponds to −1. The extension is stopped when the sum falls below $X_{max} - X_d$. In Figure 5, this process is shown for two different sequence pairs.

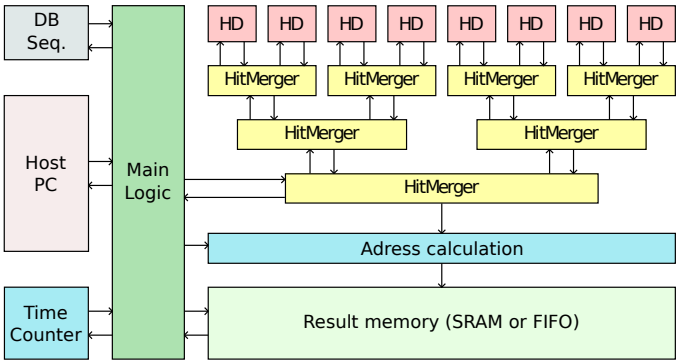

Figure 8. Global structure of the tree-like architecture for the acceleration of the BLAST algorithm on a FPGA.

Step 5: Output generation

In this step, the local alignments between the query- and the database-sequence are sorted by relevance and stored in a list.

III. Implementation

In this section, an implementation of the first two steps of the BLAST algorithm on a FPGA is described. These steps have been chosen because in sum they convey the majority of the computational effort, as has been shown by Cameron et al. [9]. The implementation is based on a tree-like architecture for the parallel extraction and combination of local alignments between a query- and a database-sequence (see Figure 8).

Due to restrictions of the current implementation, the scoring-function σ is realized as equality-function:

$$\sigma(\alpha \rightarrow \beta) = \begin{cases} True, Hit & \text{for } \alpha = \beta \\ False, noHit & \text{for } \alpha \neq \beta \end{cases} \quad (6)$$

 ISBN: 978-1-61208-585-2

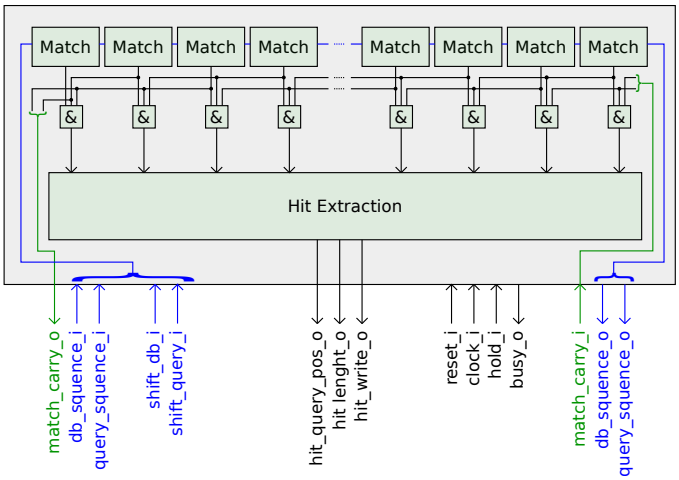

Figure 9. Structure of a hit-detector.

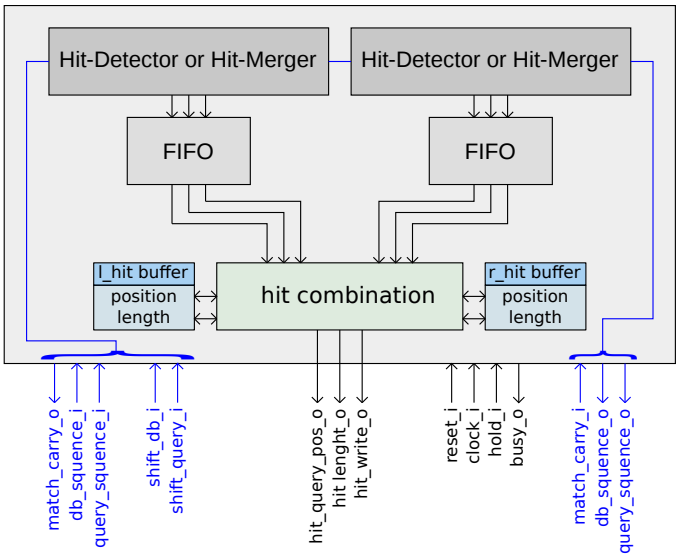

Figure 10. Structure of a hit-merger.

The maximal size of a sequence-element is five bit because a protein can have only up to 21 different amino acids. An empty or uninitialized element is described by the vector "00000".

The *Time Counter* (see Figure 8) is used for benchmark purposes only. The database-sequence is stored in the *DB Seq.* memory. The query-sequence is stored directly in the leaves of the tree-like structure. The rest of the blocks is described in the following subsections.

A. Hit-detector

The hit-detectors (*HD* in Figure 8) are the leafs of the tree. As shown in Figure 9, a hit-detector consists of a chain of n hit-matchers and a hit-extractor. The hit-matchers are basically arranged as two parallel shift registers – one for the query-sequence and one for the database-sequence – that can be shifted independently. Furthermore, they contain the logic for the scoring-function σ. The q-words introduced above are generated by AND gates of the width q.

First, the query sequence is loaded to the hit-matches. Then, the hit-extraction is started by shifting the database-sequence into the hit-matchers. The hit-extractor stores the positions and lengths of all hits that occur while shifting in the database-sequence.

B. Hit-merger

The hit-mergers (as depicted in Figure 10) are the nodes of the tree. They are used to merge overlapping hits of the previous stage in a binary tree. The previous stage can either be the hit-detectors of the first level or other hit-mergers.

The incoming hits are buffered in two FIFOs, one for the left child and one for the right child. If an entry of the left FIFO contains a right-aligned hit, the hit is stored in the left buffer. If an entry of the right FIFO contains a left-aligned hit, the hit is stored in the right buffer. If the hit combination unit detects an overlapping hit (between FIFO and FIFO or FIFO and buffer), the hits are merged and sent to the next level of hit-mergers or the result memory. Hits that can not be merged (i.e., do not overlap) are sent directly to the next level.

The length of the FIFOs depends on the hit-detector width w, the width of the q-words q and the current level in the tree th as follows:

$$f = \frac{w}{q+1} \cdot 2^{(th-1)} \tag{7}$$

The word-width of the FIFOs depends on the hit-detector width w, the current level in the tree th and the maximum query-length l_{max} as follows:

$$width = \mathrm{ld}(w) + \mathrm{ld}(th-1) + \mathrm{ld}(l_{max}) \tag{8}$$

IV. Results

In this section, the FPGA based implementation presented above is compared to an implementation in software and the impact of different parameters (i.e., query length, q-word size and hit-detector width) is evaluated. The FPGA based implementation is executed on an "Altera DE2 Development and Education board" with a clock of 50 MHz. The board contains a "Cyclone II 2C35" FPGA with 33 216 logic elements and 483 840 total RAM bits. The software implementation is executed on a Personal Computer (PC) with an Intel Core i5 Central Processing Unit (CPU) at 2.80 GHz and 8 GB Random Access Memory (RAM).

Variation of the query length: Table I shows a comparison of the computation times between PC and FPGA under variation of the query length. For all query-sequences, the same database-sequence with a length of 1 813 bases is used. The computation time of the software implementation is proportional to the query length. In contrast, the computation time of the FPGA implementation is rising much slower in relation to the query length. This is because the hits are combined in a binary tree, resulting in logarithmic characteristics.

TABLE I. Comparison of the computation times between PC and FPGA under variation of the query length.

Query length	Hits	PC / ms	FPGA / ms	FPGA / clock cycles
8	149	12.855	1.201	60093
32	786	55.155	1.219	60974
64	1598	94.331	1.246	62304
128	3050	195.607	1.296	64791
256	6263	316.306	1.406	70310
512	12081	640.919	1.611	80594
1024	23871	1241.095	2.006	100326

 ISBN: 978-1-61208-585-2

Variation of the q-word size: Table II shows a comparison of the computation times between PC and FPGA under variation of the q-word size. For all calculations, the same database-sequence and query-sequence are used. They have a length of 1 813 bases and 32 bases. Both, the PC implementation and the FPGA implementation show only little variance of computation time in respect to the q-word size.

TABLE II. COMPARISON OF THE COMPUTATION TIMES BETWEEN PC AND FPGA UNDER VARIATION OF THE Q-WORD SIZE.

q-Word size	Hits	PC / ms	FPGA / ms	FPGA / clock cycles
1	10844	47.630	1.288	64419
2	2866	47.009	1.230	61517
3	786	40.605	1.219	60974
4	199	52.516	1.217	60899
5	51	36.553	1.217	60888

Variation of the hit-detector width: Table III shows a comparison of the computation times between PC and FPGA under variation of the hit-detector width. For all calculations, the same database-sequence and query-sequence are used. They have a length of 1 813 bases and 127 bases. The hit-detector width has a high impact on the computation time of the FPGA implementation. This is because the results are processed sequentially in the hit-matchers. A lower hit-detector width results in more levels of hit-mergers and therefore a more parallel calculation. However, the global amount of necessary FIFO-memory is increased by lowering the hit-detector width. This is limiting the possible degree of parallelization, especially for large query-sequences.

TABLE III. COMPARISON OF THE COMPUTATION TIMES BETWEEN PC AND FPGA UNDER VARIATION OF THE HIT-DETECTOR WIDTH.

Hit-detector width	Hits	PC / ms	FPGA / ms	FPGA / clock cycles
4	3029	152.202	0.233	11897
8	3029	152.142	0.392	19630
16	3029	152.136	0.681	34086
32	3029	152.132	1.286	64295
64	3029	152.148	2.523	126160

V. CONCLUSION AND FUTURE WORK

It has been shown that the first two stages of the BLAST algorithm can be implemented efficiently in a parallel tree-like structure, even on a low-cost FPGA. However, for large nucleotide sequences the hit-detector width is limited to a lower bound, due to the available amount of fast on-chip memory. Of course, the hit-detector width can be increased to support larger sequences, but the result is an increased computation time.

In future work, the steps three and four of the BLAST algorithm could be implemented and evaluated according to the first two steps, to enable a complete calculation on the FPGA and thereby remove the overhead for communication. Additionally, the current implementation is not fully utilizing the available hardware, because the analysis of the next database-query is started only after the previous analysis is complete. It should be possible to decrease the computation time by starting the next analysis as soon as the FIFOs of the next level of hit-mergers are empty. However, a complex control logic would be necessary to prevent collisions and to attribute the results. Finally, an implementation of a more advanced scoring-function like the BLOSUM or PAM matrix would increase the quality of the results.

REFERENCES

[1] National Center for Biotechnology Information (NCBI), "GenBank and WGS statistics," http://www.ncbi.nlm.nih.gov/genbank/statistics/, accessed: 30. July 2017.

[2] National Human Genome Research Institute (NHGRI), "The cost of sequencing a human genome," https://www.genome.gov/sequencingcosts/, accessed: 30. July 2017.

[3] 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, 1990, pp. 403–410.

[4] M. Gokhale et al., "Building and using a highly parallel programmable logic array," IEEE Computer, vol. 24, no. 1, Jan. 1991, pp. 81–89.

[5] M. R. Mahmoodi, H. Nikaein, and Z. Fahimi, "A parallel architecture for high speed BLAST using FPGA," in 2014 22nd Iranian Conference on Electrical Engineering (ICEE), May 2014, pp. 57–61.

[6] M. Yoshimi, C. Wu, and T. Yoshinaga, "Accelerating BLAST computation on an FPGA-enhanced PC cluster," in 2016 Fourth International Symposium on Computing and Networking (CANDAR), Nov 2016, pp. 67–76.

[7] "Accelerated BLAST performance with Tera-BLAST™: a comparison of FPGA versus GPU and CPU BLAST implementations," TimeLogic biocomputing solutions, Tech. Rep., 2013.

[8] P. Evripidou and C. Kyriacou, "Data-flow vs control-flow for extreme level computing," in 2013 Data-Flow Execution Models for Extreme Scale Computing, Sept 2013, pp. 9–13.

[9] M. Cameron, H. E. Williams, and A. Cannane, "Improved gapped alignment in BLAST," IEEE/ACM Transactions on Computational Biology and Bioinformatics, vol. 1, no. 3, July 2004, pp. 116–129.

 ISBN: 978-1-61208-585-2