Acta Univ. Sapientiae, Informatica **7,** 2 (2015) 151–185

DOI: 10.1515/ausi-2015-0017

Simple scalable nucleotic FPGA based short read aligner for exhaustive search of substitution errors

Péter FEHÉR
Eötvös Loránd University
email: thepetest@gmail.com

Ágnes FÜLÖP
Eötvös Loránd University
email: fulop@caesar.elte.hu

Gergely DEBRECZENI
Wigner Institute
email: gergely.debreczeni@cern.ch

Máté NAGY-EGRI
Wigner Institute
email: nagy-egri.mate@wigner.mta.hu

György VESZTERGOMBI
Wigner Institute and Eötvös Loránd University
email: veszter@rmki.kfki.hu

Abstract. With the advent of the new and continuously improving technologies, in a couple of years DNA sequencing can be as commonplace as a simple blood test. The growth of sequencing efficiency has a larger exponent than the Moore's law of standard processors, hence alignment and further processing of sequenced data is the bottleneck. The usage of FPGA (Field Programmable Gate Arrays) technology may provide an efficient alternative. We propose a simple algorithm for DNA sequence alignment, which can be realized efficiently by nucleotic principal agents of Non.Neumann nature. The prototype FPGA implementation runs on a small Terasic DE1-SoC demo board with a Cyclone V chip. We present test results and furthermore analyse the theoretical scalability of this

Computing Classification System 1998: I.2.1, D.1.3
Mathematics Subject Classification 2010: 92D20
Key words and phrases: FPGA, parallel computing, DNA sequent

system, showing that the execution time is independent of the length of reference genome sequences. A special advantage of this parallel algorithm is that it performs exhaustive search producing all match variants up to a predetermined number of point (mutation) errors.

1 Introduction

Revolution in microbiology and genetics are producing incredible amount of data which calls for the application of the most modern tools of informatics [1]. This may include analysis of sequences from related organisms, or from apparently unrelated species. In order for a geneticist to perform analyses on genomic data it has to be obtained through a process called genome sequencing. This task can be broken down into two main sub-processes, the first of which involves extracting raw data from a sample using various instruments and the second is short read alignment, which is a purely computational problem. Recently, several short read alignment applications have been developed. The state of the art short read aligners (e.g. Bowtie[7], nvBowtie[17]) use the the Burrows-Wheeler transform[4].

The problem is the following: For every short read find the position where it best matches the reference, i.e. can be aligned to the reference sequence with the lowest number of differences. Differences can be insertions, deletions or substitution errors (also called point mutation errors). Insertions and deletions are commonly referred to as indels.

In practice the percentage of substitution errors is much higher than that of indels, therefore in this article we shall concentrate on the algorithms dealing with only substitution errors. In special cases it will be specified if indels are also taken into account.

Reference sequences are publicly available for a wide variety of species including the human genome. Sometimes shorter fragments are used instead of whole genomes in order to narrow down the search space and speed up the process. Short read data is generally produced using special sequencing instruments such as the Illumina HiSeq X Ten but it is also possible to find existing data from previous experiments using public databases.

In practice there can be more than one reference sequence, which could for example belong to different chromosomes of an organism. However, this doesn't alter the theoretical nature of the problem. So in our demonstration we decided to use a single reference sequence as our input.

In the Section (2.1), (2.2) we reviewed the numeric algorithms of DNA sequencing as Smith Waterman scosing system, Burrows-Wheeler transform.

The basic idea of our model is introduced in the Section (2.3). The Local Boolean alignment algorithms contains the Sliding windows sequential algorithm in the Subsection (3.1), Coarse grain K parallelism in the Subsection (3.2), Fine grain N-parallelism in the Subsection (3.3), these are close to hardware application. In the Section (4.) we introduced the Nucleotic algorithms, which is treating big-data strongly parallel on scalable way to realise by FPGA. This is an effective method for DNA sequence alignment. In the Section (5.) the FPGA implementation is shown on DE1-SoC Board to compare with GPU. In the Section (5.5) we presented our method in the case of Lambda virus. We compared these results with Bowtie method using a random and real sample string of Lambda phage. The correlation coefficients show significant difference between the faul and exacting matching result.

2 Numeric transformation algorithms

Traditionally the computers with the standard CPUs are ideal to perform formula calculations therefore the alignment algorithms usually applied some numerical or analytic transformation to the data which provided some computational procedure to reach the desired result. The sequence alignment in these algorithms can be applied locally or globally. The Smith-Waterman algorithm, the Burrows-Wheeler Transformation and circular convolution algorithms all perform local alignments. The global approach shown by Figure 1 is used for comparing sequences of similar length and is not discussed in this article.

```
Global: CGCGGAATGTACGATA
        CGC--AAT-TAC-ATA

Local:  CGCGGAATGTACGATA
        ---GGAAT-TACGA--
```

Figure 1: Global and local alignment

Another important concept must be introduced before moving on to the discussion of various algorithms. The DNA molecule consists of two strands: 5'→3'(forward) and 3'→5'(reverse). On the reverse strand every letter is determined by the forward strand letter in the corresponding position and vice versa (A is opposite of T and C is opposite of G). It is possible to ensure that the sequencing instruments always reads in the 5'→3' direction but it can hap-

pen on both the forward strand and the reverse strand. The direction of the reference sequence is regarded as the forward strand direction. Since a short read could have originated from the reverse strand a special transformation must be performed in order to create a version that is suitable for alignment on the forward strand. This transformed version is essentially the reverse of the original sequence after replacing each letter with its opposite. Thus for every short read we must run our alignment algorithm for the original version plus this new version called the reverse complement. This is illustrated in Figure 2.

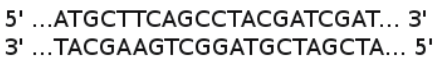

Figure 2: Forward and reverse strands of DNA

2.1 Smith-Waterman scoring system

The **Smith-Waterman algorithm**[10] performs local sequence alignment; that is, for determining similar regions between two strings or nucleotide or protein sequences. Instead of looking at the total sequence, the Smith-Waterman algorithm compares segments of all possible lengths and optimizes the similarity measure using the H scoring matrix. Backtracking starts at the highest scoring matrix cell and proceeds until a cell with score zero is encountered, yielding the highest scoring local alignment.

The basic element of this universal method which is able to deal simultaneously with substitution, point errors and indels is the calculation of the $H(i,j)$ scoring matrix:

$H(i,0) = 0$, $1 \le i \le m$, $H(0,j) = 0$, $1 \le j \le n$

$$H(i,j) = max \begin{pmatrix} 0 & \\ H(i-1,j-1) + s(a_i, b_j) & \text{Match/Mismatch} \\ max_{k \ge 1} H(i-k,j) + W_k & \text{Deletion} \\ max_{l \ge 1} H(i,j-l) + W_l & \text{Insertion} \end{pmatrix}$$

where $1 \le i \le m$, $1 \le j \le n$

We use the next notation:

a, b = Strings over the alphabet , $m = length(a)$, $n = length(b)$,

$s(a,b)$ is a similarity function on the alphabet W_i is the gap-scoring scheme.

We show one example:

Sequence 1 = ACACACTA Sequence 2 = AGCACACA

$$s(a,b) = \begin{cases} +2, & \text{if } a = b \text{ (match)} \\ -1, & \text{if } a \neq b \text{ (mismatch)} \end{cases}$$

and $W_i = -i$.

The H matrix is the following

$$H = \begin{pmatrix} & - & A & C & A & C & A & C & T & A \\ - & \mathbf{0} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ A & 0 & \mathbf{2} & 1 & 2 & 1 & 2 & 1 & 0 & 2 \\ G & 0 & \mathbf{1} & 1 & 1 & 1 & 1 & 1 & 0 & 1 \\ C & 0 & 0 & \mathbf{3} & 2 & 3 & 2 & 3 & 2 & 1 \\ A & 0 & 2 & 2 & \mathbf{5} & 4 & 5 & 4 & 3 & 4 \\ C & 0 & 1 & 4 & 4 & \mathbf{7} & 6 & 7 & 6 & 5 \\ A & 0 & 2 & 3 & 6 & 6 & \mathbf{9} & 8 & 7 & 8 \\ C & 0 & 1 & 4 & 5 & 8 & 8 & \mathbf{11} & \mathbf{10} & 9 \\ A & 0 & 2 & 3 & 6 & 7 & 10 & 10 & 10 & \mathbf{12} \end{pmatrix}$$

The alignment is reconstructed as follows: one is starting with the highest value stepping toward the next highest. A diagonal jump implies there is an alignment (either a match or a mismatch=point error). A top-down jump implies there is a deletion. A left-right jump implies there is an insertion.

For the example, the results are:

Sequence 1 = A-CACACTA

Sequence 2 = AGCACAC-A

The motivation for local alignment is the difficulty of obtaining correct alignments in regions of low similarity between distantly related biological sequences, because mutations have added too much 'noise' over evolutionary time to allow for a meaningful comparison of those regions. Local alignment avoids such regions altogether and focuses on those with a positive score, i.e. those with an evolutionary conserved signal of similarity.

The Smith-Waterman algorithm is fairly demanding of time: To align two sequences of lengths m and n, $O(mn)$ time is required. Smith-Waterman local similarity scores can be calculated in $O(m)$ (linear) space if only the optimal alignment needs to be found, but naive algorithms to produce the alignment require $O(mn)$ space.

2.2 Burrows-Wheeler Transform

The Burrows-Wheeler transform (BWT)[4] is applied on blocks of input data (symbols). It is usually the case that larger blocks result in greater compress-

ibility of the transformed data at the expense of time and system resources.

One of the effects of BWT is to produce blocks of data with more and longer 'runs' (= strings of identical symbols) than those found in the original data. The increasing the number of these 'runs' and their lengths tends to improve the compressibility of data.

The first step of BWT is to read the T string in a block of N symbols.

The second step is adding a $ character as ending symbol assigning the lowest character value to it in the alphabetic order.

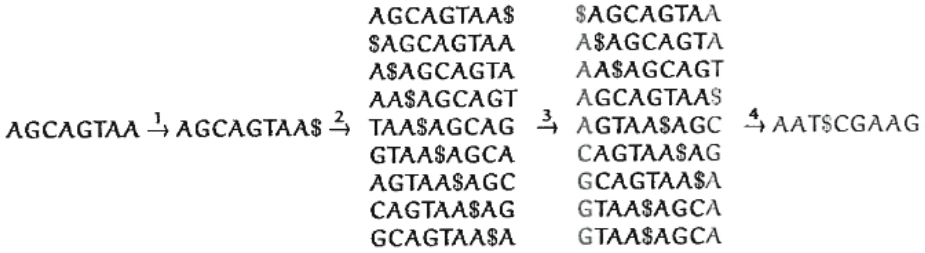

Figure 3: Burrows-Wheeler transformation steps, where red letterr are noted by F, and green characters are correspond to green L

The next step is to think of the block as a cyclic buffer: N strings (rotations). The rotation matrix may be constructed in such a way, containing the shifted blocks line by line.

The fourth step of BWT is to lexicographically sort the matrix lines (Figure 3). The first column of the matrix is denoted by F, the last column L is defined to be the Burrows-Wheeler transform of T:

$$\mathsf{L} = \mathsf{BWT}(\mathsf{T}).$$

In short:

$$\mathsf{T} = \mathsf{AGCAGTAA} \rightarrow \mathsf{AGCAGTAA\$} \rightarrow \mathsf{L} = \mathsf{AAT\$CGAAG} \rightarrow \mathsf{F} = \mathsf{\$AAAACGGT}$$

It is a very remarkable mathematical fact that knowing only L one can restore uniquely the original T string.

The first step of the reversing process is that one creates F from L by lexicographical ordering.

The basic trick of the reverse transform is the Last-to-First-Mapping prop-

erty of the L and F strings.

$$\mathsf{L} \rightarrow \mathsf{F}$$

$$\begin{array}{l} \mathsf{A} \leftarrow \$ \\ \mathsf{A} \leftarrow \mathsf{A} \\ \mathsf{T} \leftarrow \mathsf{A} \\ \$ \leftarrow \mathsf{A} \\ \mathsf{C} \leftarrow \mathsf{A} \\ \mathsf{G} \leftarrow \mathsf{C} \\ \mathsf{A} \leftarrow \mathsf{G} \\ \mathsf{A} \leftarrow \mathsf{G} \\ \mathsf{G} \leftarrow \mathsf{T} \end{array}$$

Each element of F is pointing to the symbol of L which is preceding it in T, i.e. one has from the beginning **the pair wise reconstruction of T** in the $\mathsf{L}(\mathtt{i})\mathsf{F}(\mathtt{i})$ combinations. Thus one needs only to connect them in right order.

The symbols of the T string are produced in reverse order which means that one should start from the ending character $.

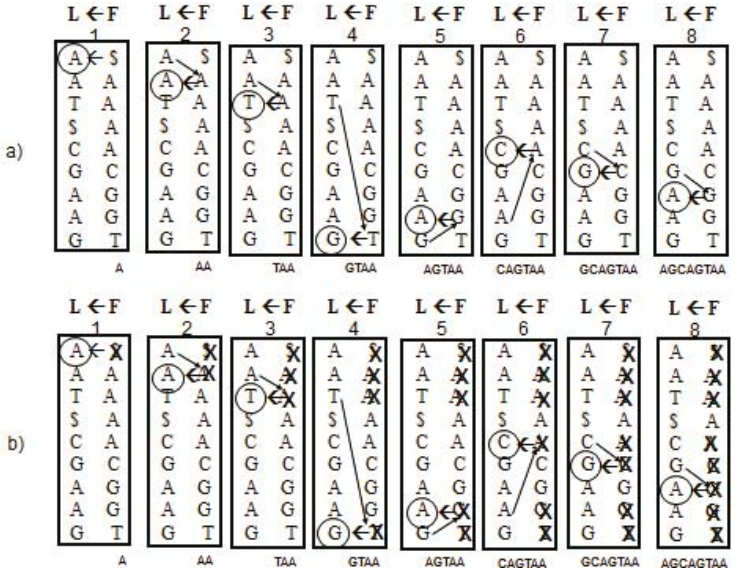

Figure 4: BWT reverse, part b) with X in F for used characters

It is worth to mark the already selected pairs in F with an 'X' as-shown in Figure 4 (b).

The horizontal arrow in the first line from F to L provides the N-th, i.e. the last symbol of T which is A.

In the next step one is jumping from the last L position (A in line #1) to the nearest F position containing the same symbol, which ensures the piecewise continuity. Thus in the above example one is ending in the second line of F.

The next horizontal arrow from F to L gives symbol A, which is really identical with the $(N-1)$-th character of T.

The procedure is repeated from the second line of L, selecting the nearest A in F which is not in the same line. Thus we reach the A character in line #3 of F.

And so on one can repeat the horizontal $F \rightarrow L$ and inclined $L \rightarrow F$ steps until one gets the final AGCAGTAA (Figure 4).

Of course, the procedure can be formulated in a more exact way too, it is based on two tables. The first is giving

Number of Preceding Symbols Matching Symbol in Current Position in L;

the second one is derived from F:

Number of Symbols Lexicographically Less Than Current Symbol

which are described in detail in ref [Burrows-Wheeler Transform Discussion and Implementation, talk by Michael Dipperstein [13]]

How can one use BWT for alignment of short reads? One can prepare the BWT of the known reference sequence containing of N characters. If the short read with m characters is identical with some part of the reference sequence, then one can assume that using the last character of the short read as starting character in F one can execute a reverse transformation from this point. As a simple test we can check in the above T as reference whether it contains the CAG combination.

In general there is not a single solution. E.g. one finds 2 combinations for CAG (Figure 5).

Try to find ACAG short read or TCAG. No way, because from the last position where the C was found one cannot go further. Let us assume that due to a point error the short read was recorded by a point error as ATCAGTAA, which has no exact matching with the reference sequence. In this case one can use some kind of heuristic method, the so called backtracking. In case of unsuccessful search the program executes some backward steps and the recorded character is changed to a new one. The selection of the position and value of the new character is depends on the measured quality of the recorded characters which is monitored during the measuring process. It is important to remark, that if one can execute exhaustive research for point error cases, then one doesnt need to apply such heuristic algorithms, which can be demonstrated

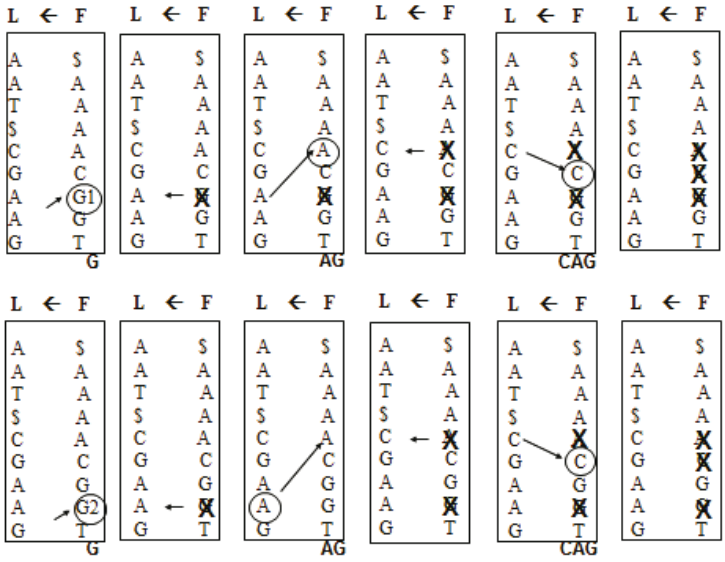

Figure 5: BWT search for partial string 'CAG', two possible solution

by the following algorithm.

2.3 Pseudo-binary circular convolution

The circular convolution is a frequently used reduced version of the general convolution formula. One can define it in the following way:

$$y(n) = h(n)@u(n) = \sum_{i=0}^{N-1} h(i) \cdot (u(n-i))_N,$$

or:

$$y(n) = h(n)@u(n) = \sum_{i=0}^{N-1} h(i) \cdot (u(n+i))_N,$$

where: $(u(n))_N$,N-point periodic extension of u(n). 'Cyclic'='circular'.

Order: 'N-point' or 'order N', $y(n)$; $h(n)$; $u(n)$ all have length N.

Here we want to specialize it further to accommodate the DNA alignment case. It will be assumed that the $u(n)$ function will correspond to the reference genome sequence of length N, whereas the short reads will be represented by $h(i)$ having non-zero values only for $0 \leq i \leq m-1$, where $m < N$, $h(i)$ is

equal to zero above this value till $i = N$. It is assumed that the characters of u and h are written in binary form of 1s and 0s, thus the total length will be increased from N to $N_b = n_{bit} \cdot N$, where n_{bit} is the number of bits required to identify the character symbols.

This binary circular convolution can have a special physical meaning if one applies and additional trick by converting the zero values to -1 in u and h functions.

Example:

$N = 5$, $m = 4$ and $n_{bit} = 2$

Symbols: a, b, c, d binary representation:	00,	01,	10,	11
pseudo binary:	-1-1,	-1 1,	1 -1,	11

Reference string: u bacad → binary:	0	1	0	0	1	0	0	1	1	1
bacad → pseudo-binary:	-1	1	-1	-1	1	-1	-1	1	1	1
Short read string: h acad → binary:	0	0	1	0	0	1	1	1	0	0
acad → pseudo-binary:	-1	-1	1	-1	-1	1	1	1	0	0

padding zeros at the end.

Binary convolution:

$$
\begin{aligned}
y(0) &= h(0)\cdot u(0) + h(1)\cdot u(1) + h(2)\cdot u(2) + \cdots + h(9)\cdot u(9) \\
&= 0\cdot 0 + 0\cdot 1 + 1\cdot 0 + 0\cdot 0 + 0\cdot 1 + 1\cdot 0 + 1\cdot 0 + 1\cdot 1 + 0\cdot 1 + 0\cdot 1 = 1 \\
y(1) &= h(0)\cdot u(1) + \ldots \\
y(2) &= h(0)\cdot u(2) + h(1)\cdot u(3) + h(2)\cdot u(4) + \cdots + h(9)\cdot u(1) \\
&= 0\cdot 0 + 0\cdot 0 + 1\cdot 1 + 0\cdot 0 + 0\cdot 0 + 0\cdot 0 + 1\cdot 1 + 1\cdot 1 + 0\cdot 1 + 0\cdot 1 = 3 \\
y(3) &= h(0)\cdot u(3) + \ldots \\
&\vdots \\
y(9) &= h(0)\cdot u(9) + \ldots
\end{aligned}
$$

Pseudo-binary convolution:

$$
\begin{aligned}
y(0) &= h(0)\cdot u(0) + h(1)\cdot u(1) + h(2)\cdot u(2) + \cdots + h(9)\cdot u(9) \\
&= (-1)\cdot(-1) + (-1)\cdot 1 + 1\cdot(-1) + (-1)\cdot(-1) + (-1)\cdot 1 + 1\cdot(-1) + \\
&\quad 1\cdot(-1) + 1\cdot 1 + 0\cdot 1 + 0\cdot 1 = -2 \\
y(1) &= h(0)\cdot u(1) + \ldots \\
y(2) &= h(0)\cdot u(2) + h(1)\cdot u(3) + h(2)\cdot u(4) + \cdots + h(9)\cdot u(1) \\
&= (-1)\cdot(-1) + (-1)\cdot(-1) + 1\cdot 1 + (-1)\cdot(-1) + (-1)\cdot(-1) + \\
&\quad (-1)\cdot(-1) + 1\cdot 1 + 1\cdot 1 + 0\cdot 1 + 0\cdot 1 = 8 \\
y(3) &= h(0)\cdot u(3) + \ldots \\
&\vdots \\
y(9) &= h(0)\cdot u(9) + \ldots
\end{aligned}
$$

From this example it is obvious that the pseudo binary circular convolution gives the exact number of bit matches between the reference sequence and the short read and the y index provides the position for that number of matching. Exact matching gives the value $y(n) = N_b$. It provides exhaustive search, because if there are more than one exact matching position then for all the n_i values one gets N_b.

In general, an error decreases the sum by 2, thus the number of matches is equal

$$M = (y(n) + m)/2.$$

The second remarkable feature of this formula is, that it works exactly in the similar exhaustive way, if we allow a given number of mismatching bits.

The third interesting fact is that one can speed up the calculations, which requires $N \cdot N$ steps, by using Fast Fourier Transform of h and u. The calculation time of the convolution will be reduced to $N \cdot \log(N)$ steps. In some architecture this can be the optimal solution, but in the next we propose even faster practical solutions.

3 Local Boolean alignment algorithms

From the definition it is obvious that the solution of the alignment problem does not require intense numerical calculations, therefore in the next we concentrate on the bit-level or string character manipulating algorithms which can be optimally executed in FPGA and ASIC systems.

3.1 Sliding window sequential algorithm

Let us assume that the reference genome has N base pair. One is looking for the alignment of short reads with the length of m base pairs. For simplicity, we assume that $N = K \cdot m$

The characters in reference genome and short read are compared individually within a sliding window (Figure 6). The number of sequential sliding steps is equals to $N - m + 1$.

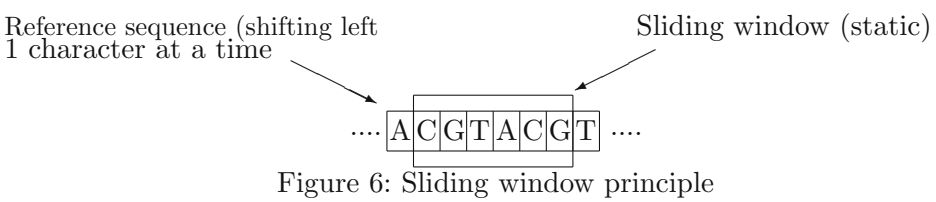

Figure 6: Sliding window principle

If one takes into account that the sequencing instruments do not give an exact copy of the measured specimen, in general there is no existing unique solution for the alignment problem. Therefore one applies statistical multiple measurement in the analysis as it is illustrated in Figure 7.

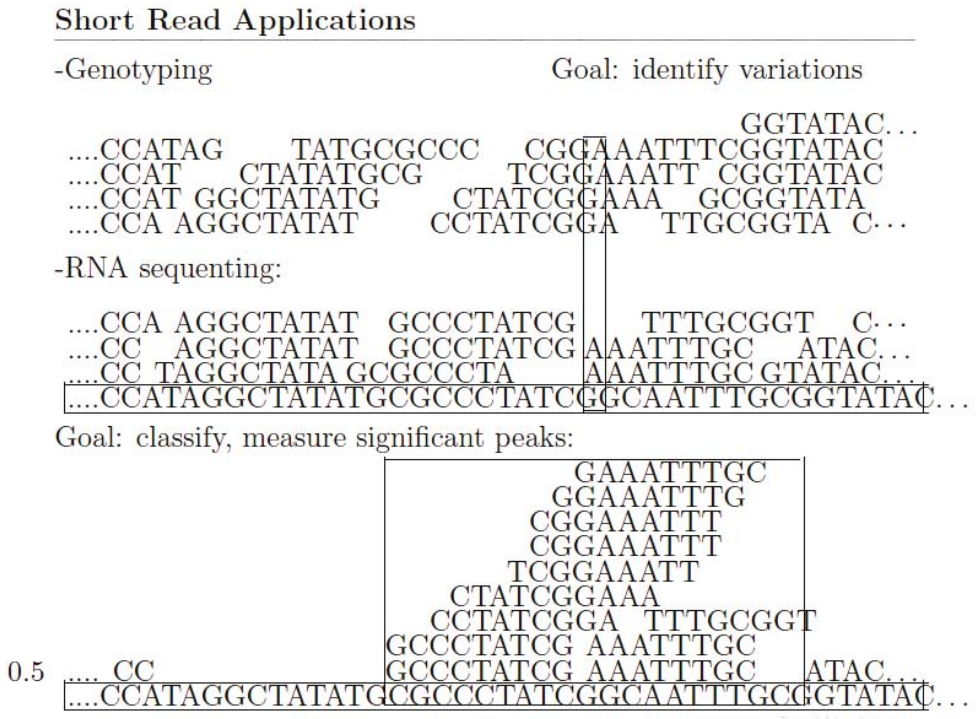

Figure 7: Statistical analysis

The base pair symbols can be converted to binary representation as it is shown in some simple examples (Figure 8).

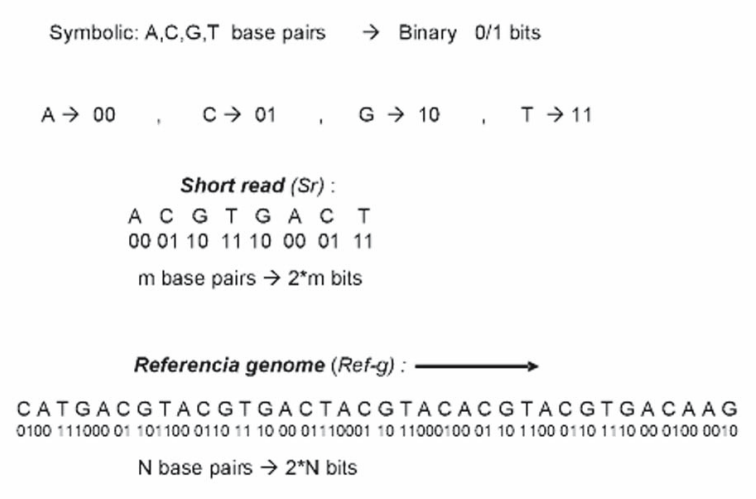

Figure 8: Illustration for symbolic binary transition

Using a single processor it takes $(N - m + 1) \cdot m$ steps to check all the combinations, which can be a very long time if N is large (Figure 9). It is not worth to compare the last $m - 1$ positions because it is not possible to have exact matching with the m-long short read.

3.2 Coarse grain K parallelism

If one applies $K = N/m$ processors then one needs only m sliding steps which reduces the execution time to $m \cdot m$ steps (Figure 10) which can be a very considerable speed up, because in general $m << N$.

The last processor will work only for the $m = 0$ case because the reference genome runs out.

3.3 Fine grain N-parallelism

If one has enough money to buy $N - m + 1$ processors then the execution will require only m steps which is very small relative to the sequential single process $(N - m + 1) \cdot m$ case (Figure 11).

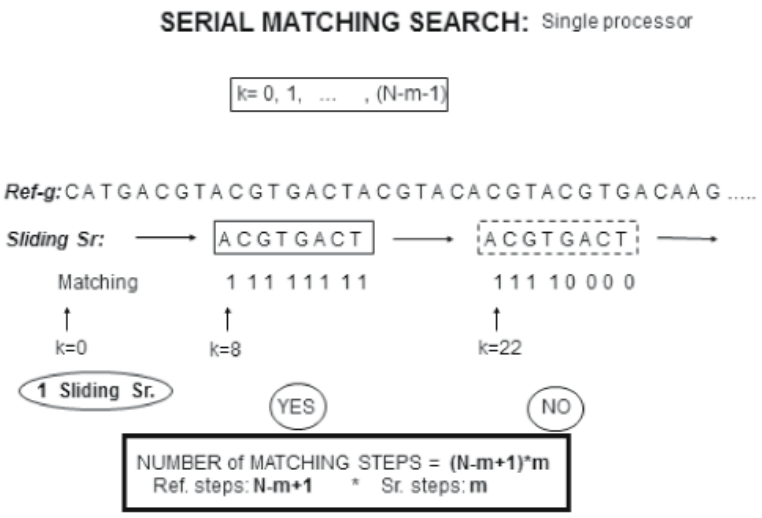

Figure 9: Single principle agent realization for serial matching search

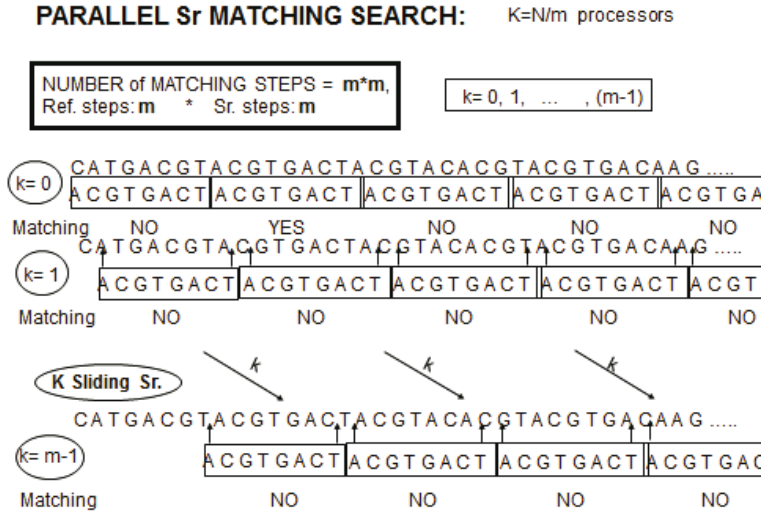

Figure 10: Moderate number of principle agents realization for coarse grain matching search

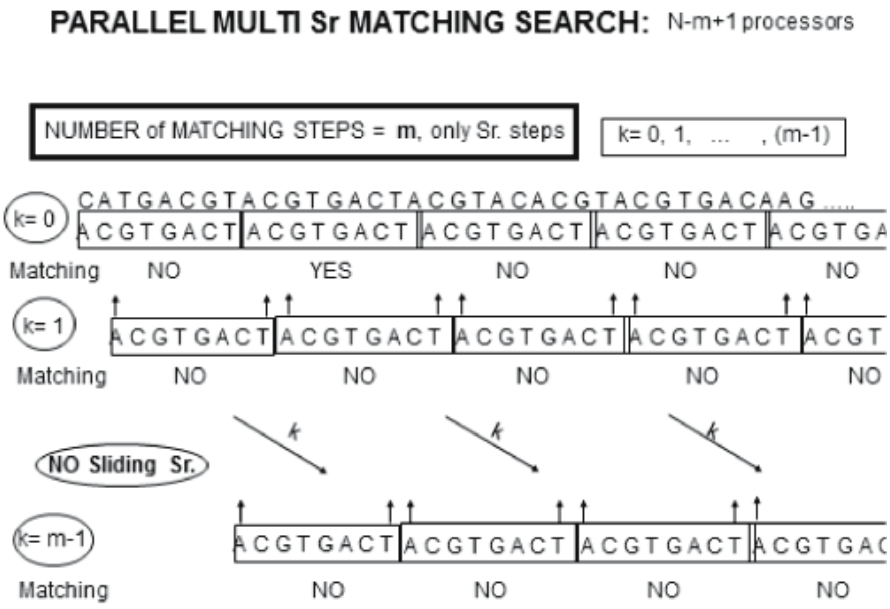

Figure 11: Fine grain matching search

4 Nucleotic algorithms

The Von Neumann architecture, also known as the Von Neumann model and Princeton architecture, is a computer architecture based on that described in 1945 a mathematician an physicist John Von Neumann and others in the First Draft of a Report on the EDVAC [9]. This describes a design architecture for an electronic digital computer with parts consisting of a processing unit containing an arithmetic logic unit and processor registers, a control unit containing an instruction register and program counter, a memory to store both data and instructions, external mass storage, and input and output mechanisms. The meaning has evolved to be any stored-program computer in which an instruction fetch and data operation cannot occur at the same time because they share a common bus. This is referred to as the Von Neumann bottleneck and often limits the performance of the system.

The proposed NON-Neumann architecture (NONN) is applying FPGA reconfigurable hardware realizing computation directly in the memory cells avoiding the CPU-memory bottle-neck. This NONN approach can be applied only for specific problems which are treating big-data massively parallel on SCAL-

ABLE way. The idea of massively parallel 1-bit CPU system is not new, e.g. a special ASIC design existed already 25 years ago [11]. The interesting fact is that large class of the presently unsolvable problems are falling in this category in physics, chemistry, biology, life sciences, materials, climate, geosciences, etc.

Exascale computing in general is a very nice idea, but in practice it seems to be a non realistic aim. Here we should like to reach this aim only in case of a limited set of problems which are important enough to be worth to invest into them. In the real physical world the matter is consisting from atoms, but the dominant element is the atomic nucleus containing 99.95 % of the mass. The solid structure of the objects is ensured by the crystal or amorphous arrangements of the ionic nuclei. The single and double helix in biological system is based on the nuclear acid base pairs. If one can follow the history of this nucleotic agents one can control the system. In wider context in cosmology stars and galaxies can play this nucleotic role. In general numerical solution of theoretical partial differential equations is achieved by discretization of space and time. The lattice nodes with definite calculation procedures can be regarded also as nucleotic objects. In a heuristic way one can define as nucleotic system those arrangements which are consisting of elements with precisely defined properties and are mainly in interaction only with other elements in their neighbourhood. This definition gives rather wide set of possibilities, the systems can have regular, amorphous, tree-like or general graph etc. structures.

According to the definition of nucleotic problems one can ensure ultrascalability, if there is a possibility to identify the so-called **principal agent** [12]. The principle agent executes the universal activity at each nucleotic site driven by the common Clock-signal. In more complex cases one can have several different types of different principal agents which are activated by special control logic at appropriate times. The procedure executed by individual principal agent can take T clock cycles corresponding to its type.

The principal agents can be regarded as vertices of a graph. The information flow is indicated by directed edges.

4.1 Bit-serial principle agent

In all the above cases each processor executes character comparison which is a two-by-two bit process, i.e. one evaluates a double-hit coincidence. Preliminary step for coincidence matrix creation in DNA principal agent is shown in Figure 12. Thus the main algorithm will work on the N*m coincidence matrix, because the two bit comparisons are restoring the symbol count length independently the coding length of the characters to N and $\mathfrak{m}$.

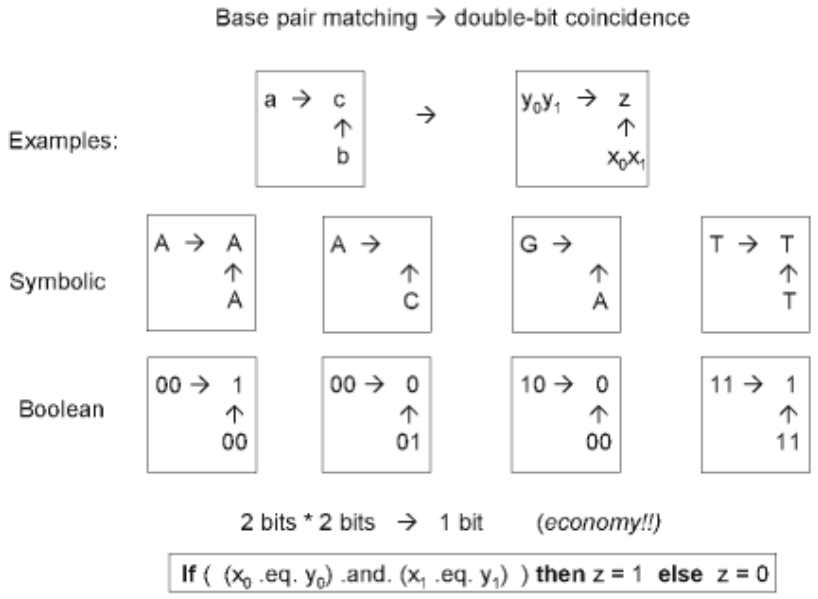

Figure 12: Preliminary step for coincidence matrix creation in DNA principal agent

The above defined single processor serial matching search algorithm can be realized in FPGA by 3 hardware elements: 2 shift-registers the first for the reference genome with length N and the second for short reads of length m, plus one principal agent.

The principal agent algorithm is extremely simple for each clock pulse the bit from reference genome is compared to the corresponding bit of the short read. The XOR logics provides output 1 in case of different inputs, thus the counter will be incremented by 1 if there was a mismatch, i.e. a point-error (Figure 13). The error counter is working in two-complement mode. Let us define the allowable maximal number of errors as Maxerr. At the start of each m-bits comparison cycle the error counter is set to -Maxerr, thus the positive value in the error counter will indicate automatically if the number of errors exceeded Maxerr.

The readout is organized through a so-called serializer to a FIFO transmitting the number of errors not exceeding Maxerr and the actual number of shifts in the reference genome to indicate the start of the matching section.

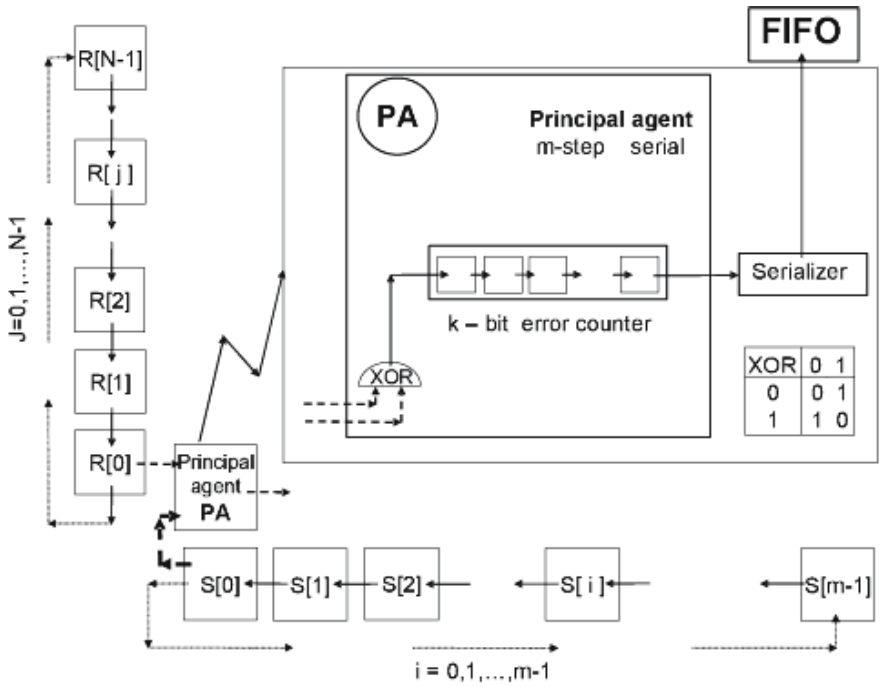

Figure 13: Bit-serial principal agent for DNA sequence alignment

One requires a serializer because due to the exhaustive search there can be more than one solutions.

In case of K principal agents the 3 main FPGA processing elements are the same. With this coarse grain design one observes a $K = N/m$ fold speed-up relative to the serial case (Figure 14). In this system each element of the reference genome is wired to one principal agent.

Here the role of serializer is more emphasized because any pair of principal agents can have simultaneous hits.

If one can afford $N - m + 1$ number of principal agents then the processing time will be independent from the length of the reference genome (Figure 15). In this system each element of the reference genome is wired to m principal agent.

It is important to remark that the 3 main FPGA elements are staying the same in all the three cases. This scaling property is an essential element of the nucleotic algorithms.

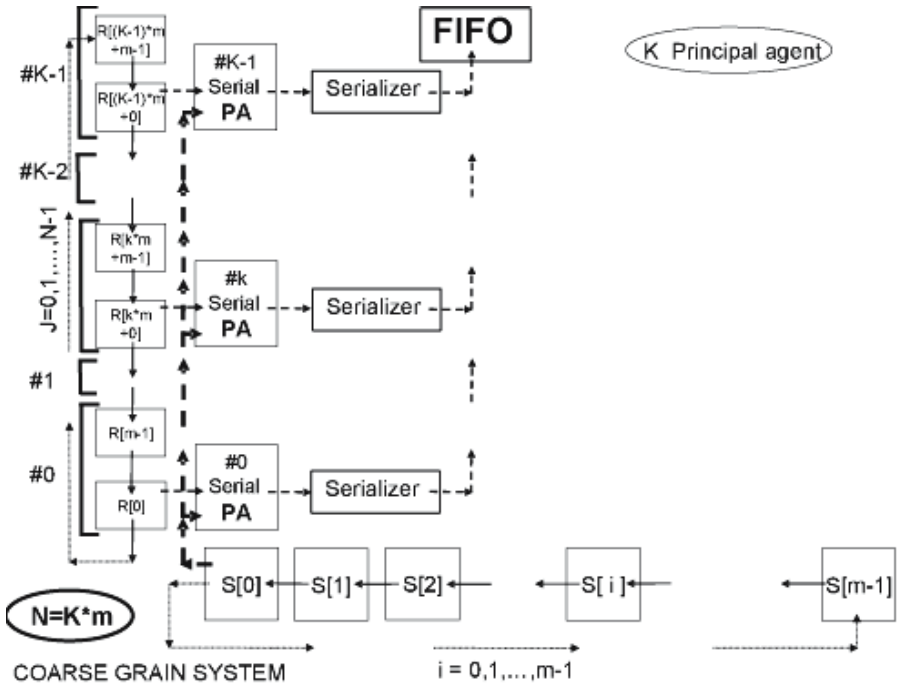

Figure 14: Coarse grain principal agents for DNA sequence alignment

4.2 ULTRASCALABILITY with bit-parallel principal agents

One can speed up the execution time and increase the efficiency of calculations by applying more complex processors. So far it was assumed that the processors were comparing one character of the reference sequence to one character of the given short read. One can perform in an FPGA (or ASIC) processor more than one comparison simultaneously.

In a special case $N = 24$, $m = 8$ and $K = 3$ the serial, coarse and fine grain systems with bit-serial principal agents are shown schematically in Figure 16.

One can apply however in the same structures instead of the bit-serial PAs so-called **bit-parallel PAs** too. In this case the exact matching can be achieved in a single clock cycle (Figure 17).

The new bit-parallel PA will have similarly simple structure, just the single XOR gate will be replaced by m pieces of XNOR gates and a m-fold AND gate to produce the exact matching trigger signal T.

This algorithm reserves the exhaustive feature of the bit-serial PA, but it

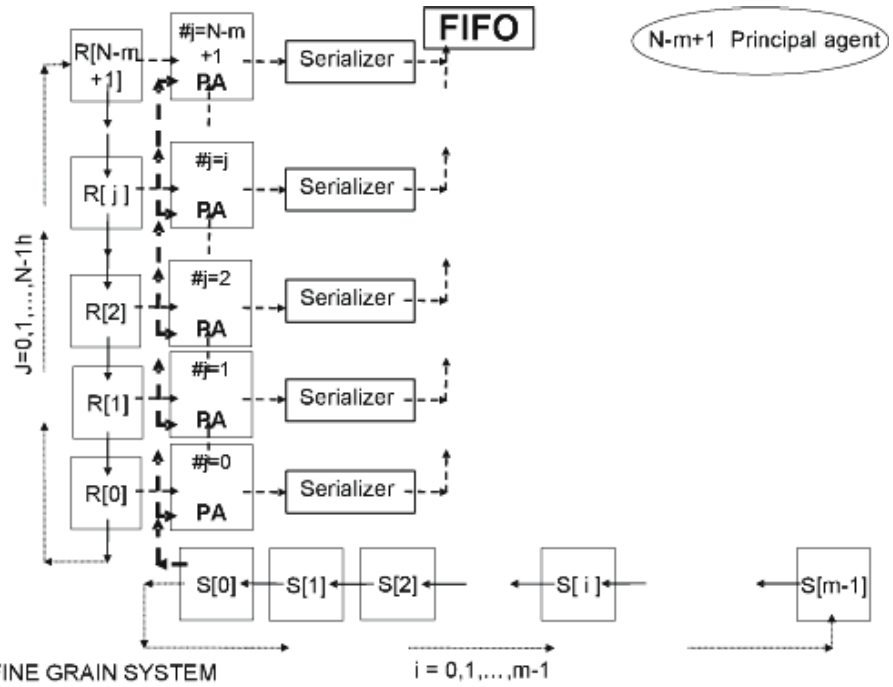

Figure 15: Fine grain principal agents for DNA sequence alignment

fails for point-errors, therefore *the speed-up was traded for performance*. One can regain part of the losses with relatively small new investments.

Let us divide the short read e.g. into $m_{reduce} = 2$ pieces and apply processors which compare m/m_{reduce} characters simultaneously. This cutting into half of the AND gate will produce m times gain is execution time and will provide extremely important information for matchings with point errors. This algorithm was proposed in [14] (Figure 18).

This simplified initial-model for DNA sequence alignment can illustrate how can one build an **ULTRASCALABLE** computer system, **where the processing time is independent from the size of the problem** if one provides the hardware which is proportional to the actual size.

In our specialized basic-model the alignment procedure can produce 3 different type of results:

a) Exact matching T=T1.AND.T2: provides the list of pointers pointing to the position of the base pair in the reference genome from where the actual short read is coinciding exactly with this part of reference genome.

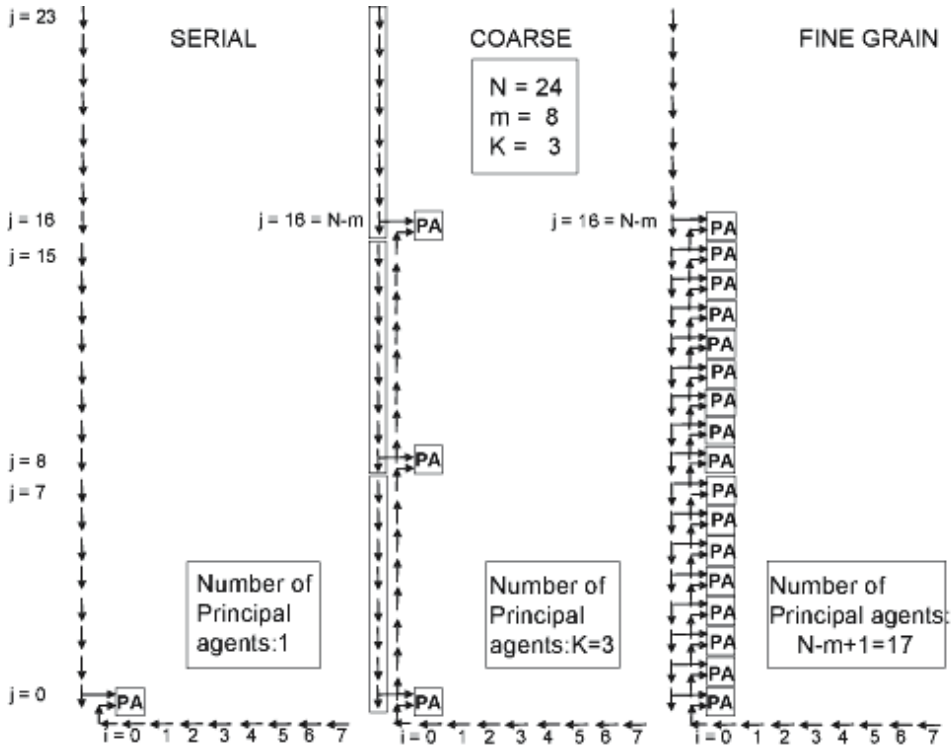

Figure 16: Summary of bit-serial principle agents

b) Half matching H=T1.XOR:T2 represents the list of exclusive OR cases, where first or second half of the short read has complete match at least of length $m/2$. If $m >> 2$ then this selection can be already very effective, therefore it is worth to sort out these cases for second part of the aligner algorithm. (If one can afford a bit more hardware for T1,T2,T3 and T4 logics then one can ensure 75% matching, allowing mismatch only in one segment shown in Figure 18 c.

c) No matching. This is the most frequent outcome. For illustration purposes intentionally we selected a combination with absolute minimal number of AND/OR gates, which simplifies the processing logics. One can easily create systems looking for more than one substitution point-errors.

One can realize the m times speed-up in both K and N parallelism.

In the K parallelism case in one clock cycle one can test the short read alignment only in those positions where the value of pointer index is a multiple

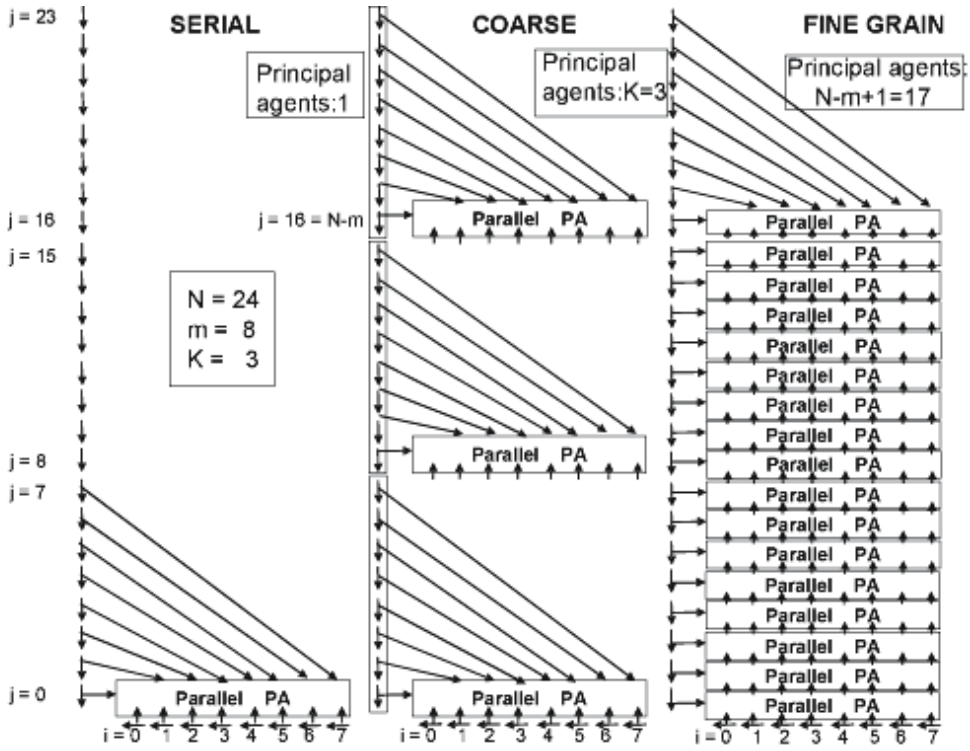

Figure 17: Summary of bit-parallel principal agents

of m, assuming that the starting index value is equal to 0. For the other values from 0 to $(m-1)$ one should make $m-1$ shifts in the reference genome and check for alignment one-by-one.

Of course, in the N parallel case one assign to each $N-m+1$ line this complex processor. Then one can get all the exact matches in single step. As additional bonus one will get a relatively short list for positions which contain all the cases where exactly one error occurred. Unfortunately there can be more than one mismatch in the indicated segment, therefore to fix this additional information a second round of tests is required. It stays however on the $O(1)$ level, because with reasonable design one can limit the expected number of multiple solutions below 10.

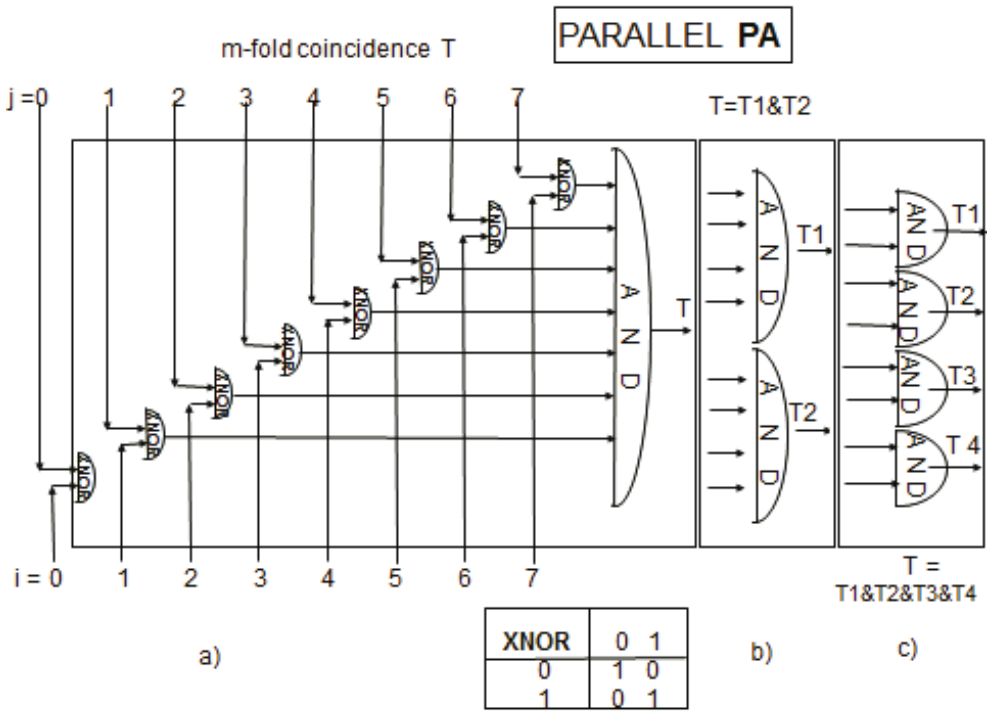

Figure 18: a) Principal agent for DNA sequence alignment with m-fold coincidence for T; b) m/2-fold coincidence + 2-fold AND for T=T1*T2; c) m/4-fold coincidence + 4-fold AND for T=T1*T2*T3*T4

5 Practical demonstration

We looked at existing FPGA-based solutions for parallel short read alignment but we did not find any that attempts ultrascalability of a system although numerous papers concluded that FPGAs provide an excellent platform on which to run sequence alignment, and that clusters of reconfigurable computers will be able to cope far more easily with the vast quantities of data produced by new ultra-highthroughput sequencers[5][6].

Some solutions use higher level languages (e.g. handel-C)[3][2] which makes them easier to implement but in most cases leads to a significant decrease in efficiency. Several papers target slow but more accurate dynamic programming approaches (e.g. Smith-Waterman algorithm)[3][6]. One particular paper[5]

discusses the implementation of a similar algorithm (Eland algorithm) on very similar hardware (DE2-SoC) but the implementation takes a more conventional approach involving hash functions and lookup tables which render ultrascalability unfeasible.

In this section we discuss the implementation of the algorithm introduced earlier. The source for the FPGA and the GPU implementation can be found in our public repository:

`https://bitbucket.org/exascalemultiscience/de1-soc-exaligner`

The proposed simple point-error search algorithm was realized in two different hardwares:

- a GPU system using CUDA with an Intel Core i7 CPU 920 2.67GHz processor and an NVIDIA GeForce GTX 980.
- an FPGA SoC(System on a chip) with a Dual-core ARM Cortex-A9 (HPS) processor and a Cyclone V SoC 5CSEMA5F31C6 Device with 85K Programmable Logic Elements

The performance as well as the result of the CUDA GPU system and the FPGA system was compared to Bowtie, a public short read aligner.

Here is a concise version of the algorithm:

```
open reference_file
while not reached end of reference_file
    reset hardware
    read reference_segment from reference_file
    write reference_segment to hardware
    open reads_file
    while not reached end of short_read_file
        read short_read from short_read_file
        write short_read to hardware
        wait until hardware is finished
        read and store results from hardware
    end while
    close short_reads_file
end while
close reference_file
open sam_output_file
```

```
for all short_read
    write alignment_data of short_read into sam_ouput_file
end for
close sam_ouput_file
```

5.1 FPGA implementation on DE1-SoC Board

The DE1-SoC Development Kit presents a robust hardware design platform built around the Altera System-on-Chip (SoC) FPGA, which combines the latest dual-core Cortex-A9 embedded cores with industry-leading programmable logic for ultimate design flexibility. Alteras SoC integrates an ARM-based hard processor system (HPS) consisting of processor, peripherals and memory interfaces tied seamlessly with the FPGA fabric using a high-bandwidth interconnect backbone. The DE1-SoC development board includes hardware such as high-speed DDR3 memory, video and audio capabilities, Ethernet networking, and much more.The DE1-SOC Development Kit contains all components needed to use the board in conjunction with a computer that runs the Microsoft Windows XP or later (64-bit OS and Quartus II 64-bit are required to compile projects for DE1-SoC) [18].

The schematic diagram of the implementation can be seen in Figure 15. In this figure the principal agents are handled as separate functional elements but in reality it is much more efficient to implement them as part of a larger functional element which carries out the computations on a large array of registers in parallel. A single instance of the prinicpal agent is illustrated in Figure 19 in detail. Each principal agent has two bits for the reference nucleotide and two bits for the short read nucleotide as input. If the output bit is 1, it means there was a match. These output bits are produced in parellel and they must be processed sequentially because the FIFO has only one input. Normally this would cause a major bottleneck, however in this problem we can discard the results with a 0 value because we don't need to process mismatches at all. This is performed by the serializers. The serializers in Figure 15 fig15(!!!!!!!!!) can also be aggregated into a larger functional element. There is some communication between the serializers in order to determine which result will be propagated to the FIFO. The number of clock cycles required for processing every principal agent output is the same as the number of matches. Most of the time there will be zero or one match in the entire reference. The case of more than one match is possible but fairly rare.

In this section ranges are always inclusive unless otherwise specified. For the implementation of the FPGA design we used Qsys and the Quartus II

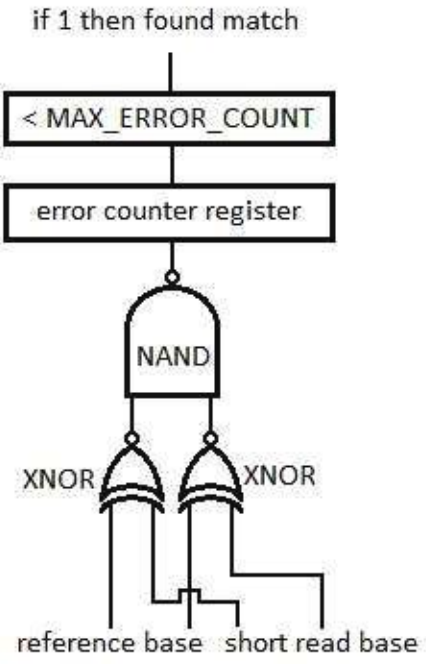

Figure 19: The single element of the fine grain principal agents

software. The HPS communicates with the FPGA fabric through a 32 bit AXI bus. The communication protocol is built according to the Avalon Memory-Mapped (Avalon MM) Interface. The interconnect between the HPS and the FPGA design is generated using the Qsys system integration tool.

Essentially, what the Qsys tool does is creating glue code, also known as interconnect, mostly consisting of buses and arbiters between the individual components of the system. For example every component has a clock input which is connected to the output of the Clock component. Another example would be the master/slave relationship between the HPS and the on-chip memory component.

There is a golden hardware reference design(GHRD) provided by the manufacturer of the DE1-SoC, Terasic. In the GHRD project there is an existing Qsys setup that was specifically designed for this device. This project perfectly matches the capabilities of the device and it can be easily extended with additional functionality.

During Avalon MM transfers the processor takes the role of the master and

the FPGA accelerator logic behaves as the slave. This means that transfers are always initiated by the C program and the FPGA design reacts to it within a few clock cycles.

In the C code Avalon MM transfers are simple read and write operations at a virtual memory address, which can be calculated by adding the appropriate offset to the virtual base memory address corresponding to the FPGA accelerator logic. The offset values and the virtual base memory addresses have to be synchronised between the Qsys setup and the C code.

From the perspective of the FPGA design the memory ranges from 0 to 3 (4x32 bits). Values in this range can be encoded using a 2-bit wide value. In the C code the virtual memory offset values range from 0x0 to 0xb (4x4 bytes).

According to the communication protocol it is the responsibility of the accelerator logic to keep track of whether the next input is part of a short read or the reference. Calculations start as soon as the loading of all necessary inputs has finished. The results are then pushed into a FIFO. The C code uses polling on a designated memory address to determine whether the FIFO holds some data, i.e. a result is available to read. Reading the actual results takes place on another memory address specifically allocated for the task. The FPGA can also be reset from the C code using an Avalon MM transfer to the appropriate memory address.

5.2 GPU implementation

In the GPU implementation most of the C code is the same as in the FPGA implementation. The main difference is in the communication between the two components (CPU and GPU). Instead of explicit transfers through an interface the CPU has indirect access to the allocated memory where the GPU calculations take place. Before and after a kernel call the data has to be copied between the CPU memory and the GPU memory. Though it is considered good practice to over-issue work to the GPU to help to the device memory latency.

From an algorithmic perspective we can observe a correspondence between principal agents in the FPGA and CUDA kernel threads. We applied the same pseudo code (Section 5.) in both cases, despite the fact that the GPU would be able to scan the entire string in the memory at one time.

5.3 Comparison

Bowtie numeric vs Boolean: There is no difficulty in finding one match if there

is any. The parallel nature of Boolean approach guarantees the exhaustive search whereas in Bowtie it would need special effort to look for multiple solutions. Of course, the existence of multiple solutions requires additional programming to serialize the candidates.

In case of the heuristic Bowtie and its more advanced versions the processing of almost good matchings with few number of errors is a real challenge. One of the possible solutions is the backtracking in BWT which means a trial and error procedure to look for exact matching artificially modifying characters in the measured short reads assuming recording errors in the given position.

GPU vs FPGA: Programming in GPU and FPGA requires completely different methodology. One should learn new languages CUDA (or OpenCL).

Nucleotic nature of the problem: extreme simple algorithms.

Hardware difference: For the FPGA minimal resources are required and calculations are executed in the memory cells, whereas in the GPU system complete threads are sacrificed, essentially a small CPU is used for each PA.

GPU threads have separate local memory and a very much reduced but still rather complex mini-CPU for real and integer operations. Few number of threads per square cm of silicon.

In FPGA memory cells and logical gates are together in logical elements ideal for bit level programming, one could say that calculation is performed inside the memory cells. The logical elements are relatively universal, but simple enough not to waste silicon surface for unused resources. One can have millions of logical elements per square cm on silicon.

The use of ASIC hardware can be even more economical. One can design only with the minimally necessary memory cells and gates, thus no unused elements are sitting in reserve. Its density can reach billions per square cm on silicon.

The scalability problem occurs for both architectures because the same signal should be delivered to more and more elements. FPGA and ASIC have a much larger density per chip resulting in shorter transmitting routes!! Furthermore the network topology of such a system can be very flexible as it can conform to the custom data-flow of any algorithm while GPUs have a fixed infrastructure for transmitting data.

In case of nucleotic problems the network consists of mainly two components:

In first case there are connections only between neighbouring principal agents. This structure is ideal for ultrascalability, because one can increase the network by simple connections at the edges.

The other part of network is given by a few global transmissions propagating information to a large number of elements.It is not a problem to send

signals on a single line to 10 destinations, but the signal propagation becomes questionable if one aims for millions of destinations with a single signal. In principle, one can use a binary tree instead of a single line with appropriate amplifying elements. Of course, the expansion of this binary distribution tree is much more complex between chips and boards than inside a chip. The same logics can be applied in GPU, FPGA and ASIC principal agents, but it is obvious that it costs practically nothing in energy and cost at ASIC, but can be prohibitive in case of GPU boards.

The I/O capabilities in FPGAs are larger by several orders of magnitude (due to the pin counts) making it easier to incorporate it into a larger system.

Longer clock cycles are a serious disadvantage in FPGA and ASIC systems but as the technology is advancing they are approaching speed of traditional CPUs. The clock frequency also depends on the timing constraints of the RTL design.

The number of FPGA processing units can not be increased further (as it was explained in Section 5.3) due to a hardware limit, over up to these values we can extrapolate, assuming the same scale behaviour. GPU have been possible to go further, but we did not want to compare the theoretical values with the measured run time. We might expect that longer chains and multi-thread processor, the GPU efficiency will grow roughly up to 4-8 times of the processing units, which reaches its peak efficiency and run time does not improve in the future.

The table below illustrates the difference between the running time of the FPGA and GPU implementations. The first column contains the number of principal agents/threads used in each run and the second and third columns contain the running times for the Lambda phage example detailed in a later section. It only takes 1024 principal agents to run faster than the GPU due to the ultrascalability property of the FPGA implementation. In the GPU implementation the more threads the longer it takes to evaluate the result of every thread because it must be performed serially by the CPU. It would be interesting to compare the two solutions using an even higher degree of parallelization but due to the limited capacity, a design with more than 1024 principal agents doesn't fit in the Altera Cyclone V FPGA.

The runtimes of introduced method on FPGA and GPU are shown in Table 1. In comparison if we run the algorithm sequentially it takes 147940.670 seconds (approximately 41 hours).

	FPGA[s]	GPU[s]
64	2427.410	2282.875
128	1214.420	1159.218
256	608.870	588.058
512	304.890	298.164
1024	154.280	157.256

Table 1: Runtime in FPGA and GPU

5.4 Simple example

In this section we describe the different implementations of the exaligner algorithm.

Running the CUDA application is extremely simple. One should execute the following command:

- exaligner-gpu example_reference.fa example_reads.fq example.sam

The FPGA application is very similar:

- exaligner-fpga example_reference.fa example_reads.fq example.sam

With Bowtie it is a little different:

- Command line for index creation:
  bowtie2-build reference/example_reference.fa example_reference
- Command line for sequence alignment:
  bowtie2 -x example_reference -U reads/example_reads.fq -S example.sam

Relevant positions in the output sam file:

Position #1.: short read ID
Position #2.: bitset, possible values in our case: 0,4,16; if 0, then forward maching; if 16, then reverse matching; if 4, then no alignment found
Position #4.: aligned reference genome position, indexing starts with 1.
Position #10.: short read sequence

The first few lines of the example.sam output file:

@HD VN:1.0 SO:unsorted
@SQ SN:example_reference1 LN:420
@PG ID:Exaligner VN:1.0 CL:'./exaligner-fpga example_reference.fa example_reads.fq example.sam'
example_read1 0 example_reference1 **5** 42 16M * 0 0 **TGATGGTCGTCCATTA** .:7@3<6&10EG2<7<
example_read2 16 example_reference1 **4** 42 16M * 0 0 **TTGATGGTCGTCCATT** <7<2GE01&6<3@7:.

Below is the first matching short read from the example_short_reads.fq file. The last row describes the sequencing quality and can be ignored.

@example_read1
TGATGGTCGTCCATTA
+
.:7@3<6&10EG2<7<

In this example the following simple reference file was used:

> example_reference1
GCCTGTATGGTCGTCCATTAAGTACGCTAAGTCACAGCGCGCTGC
GCCAGGGCGTGGCAATGGTGCAGCAAGATCCGGTGGTGCTGGCGG
ATACCTTCCTCGCCAACGTGACGCTGGCACGTGATATCTCTGAAG
AACGCGTCTGGCAGGCGCTGGAAATCGTGCAGCTGGCGGAGCTGG
CGCGTAGCATGAGTGATGGTATTTACACGCCGCTTGGCGAGCAGG
GGATAAATCTCTCAGTCGGGCAAAAGCAACTGCTGGCACTGGCGC
GCGTGCTGGTGGAGACGCCGCAAATCCTGATCCTTGATGAGGCAA
CCGCCAGCATTGACTCCGGGACTGAATAGGCGATTCAACATGCTC
TGGCGGCGGTGCGTGAACATACTACGCTTGTGGTGATTGCTCACC
GCTTATCAACTATTG

The .sam output of the FPGA and GPU applications are identical. One should not expect identical results from Bowtie because its algorithm uses heuristics and the output is random. However, as the next section describes the output files compare favourably.

5.5 Lambda virus

In this article we introduced an algorithm using the advantage of the FPGA options to determine the DNA sequences. We present below the case of Lambda virus, because the DNA molecule of 48502 base-pairs is linear.

In 1950, Esther Lederberg an American microbiologist, who performed an experiments on E.coli mixtures. His research led to employment of Lambda phage as a model organism in microbial genetics as well as in molecular genetics [8].

In this article we study the short reads each comparison performed by the reference sequence, are determined for exact matching, 1, 2, and 3 cases of error.

We ignore the indel ie. the insertion, when an extra element appears and the deletion case, when an item is missing in the test sequent.

We calculated the distribution of short reads over Lambda virus, where the length of short reads is 50. The number of short reads depends on the reference position, which was calculated by exaligner algorithm (5.5 Section). This method is able to accurately determine individual cases of error occurrences.

The generated sample file is created by wgsim program [15]. The exact matching and 1 cases of error have been shown in Figure 20. The 2 and 3 cases of error have been presented in Figure 21. The generated and real [16] sample string of Lambda phage was studied by Bowtie algorithm also, which was shown in Figure 22.

Since the set of measurements considered as random sequence, therefore we can characterize this series with the expected value, standard deviation and the correlation coefficient in the Table 2.

There are significant difference of frequency value between the faul (Figure 20 (a)) and the exact matching (Figure 20 (b)) release. The correlation coefficient changes significantly in this case.

	0	1	2	3	B-g	B-r
0	1	0.6144	0.4194	0.3792	0.4660	0.0241
1	0.6144	1	0.8215	0.7708	0.8502	0.0391
2	0.4194	0.8215	1	0.9854	0.9718	0.0218
3	0.3792	0.7708	0.9854	1	0.9612	0.0159
B-g	0.4660	0.8502	0.9718	0.9612	1	0.0227
B-r	0.0241	0.0391	0.0218	0.0159	0.0227	1

Table 2: Correlation coefficient (B-g: Bowtie alg. on generated string, B-r: Bowtie alg. on real string)

The FPGA method is reconfigurable and scalable, so this algorithm can be developed further to find the indels and more complicated and longer DNA sequence.

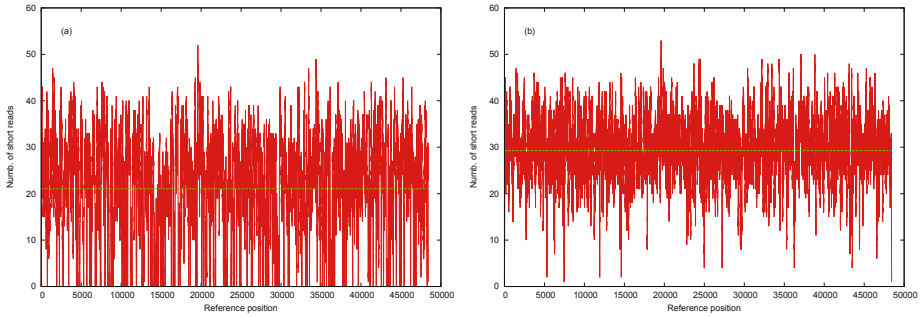

Figure 20: The diagram for DNA sequence alignment, which consist of 0 (a) resp. 1 (b) error

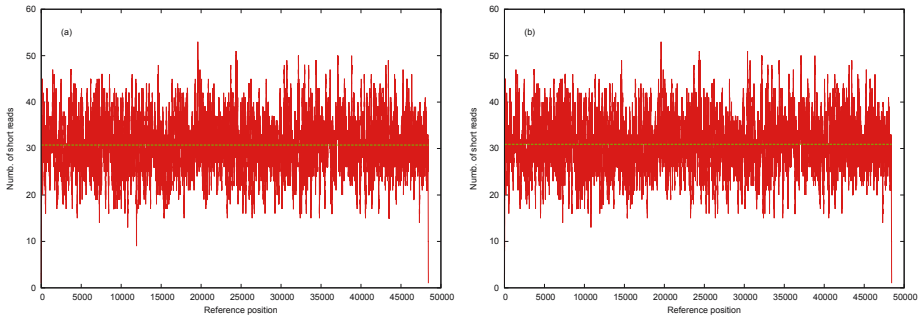

Figure 21: The diagram for DNA sequence alignment, which consist of 2 (a) resp. 3 (b) errors

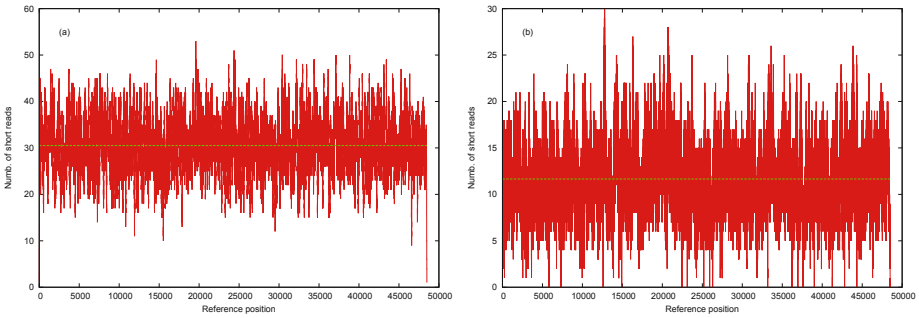

Figure 22: The diagram for DNA sequence alignment using Bowtie alg., which consist of the generated sample resp. (a) real string (b)

6 Summarize

In this article, we introduced a new nucleotid method that is suitable for processing large amounts of data, which is close to the hardware algorithms (FPGA). We are shown in case of DNA sequenceces of lambda virus to use the exaligner procedure. This method is reconfigurable and rescaling, therefore it can be developed on the more effective tools to study much larger database as the human genom sequence.

7 Acknowledgements

The authors thank prof. I. Csabai for useful discussions and valuable remarks.

References

[1] L. B. Alexandrov, Serena Nik-Zainal, et al., Signatures of mutational processes in human cancer, *Nature* **500 (7463)** (2013) 415-421. ⇒152

[2] J. Arram, K. H. Tsoi, Wayne Luk, P. Jiang, Hardware Acceleration of Genetic Sequence, Chapter: Reconfigurable Computing: Architectures, Tools and Applications, *Lecture Notes in Comp. Sci.*, **7806** 13–24. ⇒173

[3] K. Benkrid, Liu Ying, A. Benkrid, A highly parameterized and efficient FPGA-based skeleton for pairwise biological sequence alignment, very large scale integration (VLSI) systems, *IEEE Transactions* **17,** 4 (2009) 561–570. ⇒173

[4] M. Burrows, D. J. Wheeler, **124** (1994), *A block sorting lossless data compression algorithm*, Technical Report, Digital System Research Center. ⇒152, 155

[5] Y. S. Dandass, S. C. Burgess, M. Lawrence, S. M. Bridges, Accelerating string set matching in FPGA hardware for bioinformatics research, *BMC Bioinformatics* **9** (2008) 197. ⇒173

[6] R. K. Karanam, A. Ravindran, A. Mukherjee, C. Gibas, A. B. Wilkinson, Using fpga-based hybrid computers for bioinformatics applications, *Xilinx Xcell Journal* **58** (2006) 80–83. ⇒173

[7] B. Langmead, C. Trapnell, M. Pop, Sl. Salzberg, Ultrafast and memory-efficient alignment of short DNA sequences to the human genome *Genome Biol.* 10:R25., ⇒152

[8] E. Lederberg, Lysogenicity in Eescherichia coli strain K-12, *Microbial Genetics Bulletin*, **1** (1950) 5–8. ⇒182

[9] J. von Neumann, *First Draft of a Report on the EDVAC* pp. 149. University of Pennsylvania, June 30. 1945. ⇒165

[10] T. F. Smith, M. S. Waterman, Identification of common molecular subsequences, *Journal of Molecular Biology* **147** (1981) 195-197. ⇒154

[11] G. Vesztergombi, 'Iconic' tracking algorithms for high energy physics using the trax-I massively parallel processor, CHEP, *Computer Physics Communications*, **57** (1989) 290–296. ⇒166

[12] G. Vesztergombi, One billion processors program's demo on FPGA emulator board, IEEEXplore, ReConFigurable Computing and FPGAs (ReConFig), (8–10 Dec. 2014) International Conference, Cancun. ⇒166

[13] ∗ ∗ ∗ Burrows-Wheeler Transform Discussion and Implementation Homepage: http://michael.dipperstein.com/bwt/ ⇒158

[14] ∗ ∗ ∗ EXAMS project submitted to EU call: FET-Proactive – towards exascale high performance computing H2020-FETHPC-2014. Publication date 2013-12-11 Deadline Date 2014-11-25 17:00:00. Specific challenge: The challenge is to achieve, by 2020, the full range of technological capabilities needed for delivering a broad spectrum of extreme scale HPC systems. (Private communication) ⇒170

[15] ∗ ∗ ∗ Generated sample: https://github.com/lh3/wgsim. ⇒182

[16] ∗ ∗ ∗ Real Lambda phage Homepage: `ftp://ftp.ncbi.nlm.nih.gov/genomes/all/GCF_000840245.1_ViralProj14204/GCF_000840245.1_ViralProj14204_genomic.fna.gz` ⇒182

[17] ∗ ∗ ∗ NVBIO: nvBowtie, 2015,
Homepage: http://nvlabs.github.io/nvbio/nvbowtie_page.html. ⇒152

[18] ∗ ∗ ∗ Terasic – DE Main Boards, datum, Homepage: http://de1-soc.terasic.com. ⇒175

Received: September 15, 2015 • Revised: December 22, 2015