Identification of DNA Signatures via Suffix Tree Construction on a Hybrid Computing System

Lin Feng, Angela Jean, Chong Poh Leng, Lai Danbo
School of Computer Engineering
Nanyang Technological University
Singapore 637989
e-mail: asflin@ntu.edu.sg

Abstract: - Identification of DNA signatures has been empowered by the increasing availability of thousands of bacterial and viral genomes through the next-generation sequencing technologies. In exploration for the computational solution, the suffix tree has been proposed as a data structure well suited to analyzing genomic sequences because it enables the storage of long strings in a logical, indexed manner for fast retrieval. We propose a new algorithm for identification of DNA signatures, based on exploitation of the endogenous features of a genomic sequence. These features are revealed in a construction of suffix array. Furthermore, to greatly speed up the time-consuming process, the core algorithm is mapped and routed on to a Field-Programable Gate Array (FPGA) for parallel implementation. In this paper, we will focus primarily on the relationship of the data structure and the features associated. We will then describe the deployment of the hybrid computing system on a HyperTransport compliant architecture. Illustrations are always given to clarify the technical details, and experimental results are presented to verify the correctness of the algorithm.

Key-Words: DNA signature; endogenous feature; suffix tree; Field-Programable Gate Array; hybrid computing

1 Introduction

The genomics research landscape has been reshaped since the advent of high-throughput next-generation sequencing (NGS) technology [1, 2] which surpasses the old Sanger sequencing paradigm. One immediate outcome of such sequencing technology is the generation of very large amounts of data. Researchers can now compare one genome with another, or even perform meta-analyses of whole sets of genomes, instead of limited regions.

As a case study in this paper, the influenza virus genome consists of eight single-stranded RNA segments that code for eleven known proteins. The PB2, PB1, and PA segments encode the RNA polymerase, and HA, NP, NA, and M encode haemagglutinin, nucleoprotein, neuraminidase, and the matrix proteins, respectively.

There are three types or genera of influenza viruses within this family- A, B and C. Only types A and B cause significant human disease. Type A viruses are the most important pathogens of the three types and there have been several human pandemics caused by influenza A viruses. Influenza A viruses can be further categorised into subtypes according to the antigenic properties of their haemagglutinin (HA) and neuraminidase (NA) surface glycoproteins. HA is a protein that causes red blood cells to agglutinate and NA is an enzyme that is crucial for the process that is required for proper budding and release of progeny virions from the host cell surface. Presently, there are 16 known subtypes of HA and 9 known subtypes of NA. Three subtypes have been shown to cause pandemics during the last century: H1N1 caused “Spanish flu” in 1918, H2N2 caused “Asian flu” in 1957, and H3N2 caused “Hong Kong flu” in 1968. Although there have been many subtypes of influenza A reported so far, only some subtypes of influenza A virus have been associated with human infection. Some examples are H1N1, H1N2, H3N2, H5N1, H7N2, H7N3, H7N7 and H9N2.

Seasonal influenza virus infections in humans cause yearly epidemics that result in millions of human infections worldwide and have significant health and economic burdens. In Singapore, there have been more than 1,600 hospital admissions from complications related to the Influenza A (H1N1) infection since July 2009, including 20 deaths and the annual all-cause death rate from seasonal influenza in Singapore was estimated at 14.8/100,000 person-years.

Avian influenza, commonly known as bird flu, is an infectious viral disease in birds that is caused by several types of influenza viruses. Most do not infect humans. Some, however, such as H5N1, have caused serious infections in humans. Avian Influenza A viruses are categorised into two groups based on their ability to cause disease in poultry:

high pathogenicity or low pathogenicity. Highly pathogenic avian influenza (HPAI) viruses result in high death in some poultry species while low pathogenicity avian influenza (LPAI) viruses also cause outbreaks in poultry but are not generally associated with severe clinical disease. Avian influenza A virus subtypes H5N1, H7N7, and H7N3 have been associated with HPAI, and human infection with these viruses have ranged from mild (H7N3, H7N7) to severe and fatal disease (H7N7, H5N1). Human infection with LPAI viruses has been reported, including very mild symptoms (e.g., conjunctivitis) to influenza-like illness. Examples of LPAI viruses that have infected humans include H7N7, H9N2, and H7N2.

Analysis of these massive and heterogeneous data poses multiple computational challenges. These include effective data mapping, annotation and visualization; efficient data storage and retrieval, and the integration and interpretation of data from multiple technological platforms each using different sequencing chemistries, and each with its own unique output and error characteristics [18, 19, 20]. Algorithms that work well for smaller scale problems are either insufficient or inappropriate for current genomic analysis. Dynamic programming techniques, such as Smith-Waterman [3] and Needleman-Wunsch [4] algorithms work well to align regions of interest between two genes, whereas heuristics (trial-and-error) must be applied when highly conserved regions between two entire genomes are to be identified in a reasonable amount of time and space [5]. Moreover, in genomic sequences, the demarcations between functional groups of nucleotides or amino acids are more subtle, for example, the start codon determining the beginning of an open reading frame is identical in sequence to any internal methionine amino acid, and the phenomenon of alternative splicing can generate multiple transcripts from the same stretch of DNA sequence. This further increases the difficulties in aligning sequence pairs.

In exploration for the solution, the suffix tree has been proposed as a data structure well suited to analyzing genomic sequences because it enables the storage of long strings in a logical, indexed manner for fast retrieval. The concept of suffix tree was first introduced as a position tree by Weiner [6] in 1973, and the construction of suffix tree for sequence representation was further improved by McCreight [7] in 1976 and Ukkonen [8] in 1995. The suffix tree can be used to provide exact matches efficiently, which many heuristics depend on. In a nutshell, suffix tree and generalized suffix tree (the multiple string variant of suffix tree) can be used to solve a number of computational biology related problems in optimal space and time. The challenge is the computational efficiency in construction of the suffix tree.

We propose a new algorithm for identification of DNA signatures, based on exploitation of the endogenous features of a genomic sequence. These features are revealed in a construction of suffix array (an implementation of the generalized suffix tree). Furthermore, the core algorithm is mapped and routed on to a Field-Programable Gate Array (FPGA) for efficient implementation. We will focus primarily on the relationship of the data structure and the endogenous features. Illustrations will be given to elucidate the representation of the features. We will also describe the deployment of the hybrid computing system on a HyperTransport compliant architecture, and the computational experiments on identification of DNA signatures for influenza A virus.

2 Endogenous Feature Recognition

In this section, we study key endogenous features of a genomic sequence which have significant meaning in DNA signatures, including repeats, lift-diversity and those features biased towards the ends of the gene. We discuss how these features are revealed in a generalized suffix tree.

2.1 Construction of a Suffix Tree of Endogenous Feature

Sequences that are unique to a given species or strain are especially useful to distinguish the target organism from either related or un-related species. Such unique sequences are termed DNA signatures or genomic markers, and are important in areas such as the identification of genes responsible for drug resistance, accurate detection of pathogens especially those which are used as weapons of bioterrorism or warfare, and epidemiological analysis such as the identification of bacterial strains that cause food poisoning.

A naïve method of obtaining DNA signatures from a selected genomic sequence is to utilize a sliding window of length n. Starting from the first position of the target sequence and ending at the nth position, the window is moved down from the 5' end of the target sequence towards the 3' end, until the start of the window reaches the $(m-n+1)$-th position, where m is the length of the target sequence. At each position $(m-n+1)$, the feature in the window is subjected to a filtering analysis.

The reliability of a signature is dependent of its sensitivity and specificity to its genomic sequence,

and this usually involves a time-consuming and exhaustive searching in the entire genome. This problem can be streamlined by using an effective data structure to improve algorithmic efficiency. To facilitate all these time-consuming operations, we device a Suffix Tree of Endogenous Feature, or EF-Tree, to aid in the signature identification process.

An EF-Tree is a data structure derived from a generalized suffix tree (GST) [9] for representation of the features. The tree is created from a set of substrings, *S*, that represents the set of all features from a genomic sequence such as a gene. Given a sequence of length *m*, and a feature size of length *n*, a set of (*m*-*n*+1) features may be obtained. Construction of the EF-Tree is initiated with the first feature in the set *S*, as outlined by the following procedure:

Proc *create_eftree*(*S*) {
1. Initialization: T_1, an implicit tree, has one edge labeled S[1];
2. for (i = 1 to m-1) {
3. build T_{i+1};
4. for (j = 1 to i+1) {
5. if (S[j..i] ends at a leaf of T_{i+1}) {
6. Add character S[i+1] to the end of label on the edge to the leaf; }
7. elseif (no path from the end of S[j..i] location continues with S[i+1])
8. Split a new leaf edge for character S[i+1];
9. elseif (S[j..i] ends in the middle of an edge) {
10. Create an internal node;
11. Compute properties of substring from previous node to new node;}
12. else // S[j..i+1] is already in the tree
13. Continue;}}
14. Expand the final implicit tree to create a full EF-Tree;}

In procedure *create_eftree*, line 11 of the code may be implemented for calculation of various properties, such as the melting temperature of the substring from the previous node to the current node. If the previous node is the root node, then this is the melting temperature of the prefix of the suffix that is currently being added to the initial EF-Tree. Once the suffix tree of the first feature from the set *S* is constructed, subsequent features in the set can be added to the tree to create a full EF-Tree. The computed sequence values such as melting temperatures stored in the nodes can be easily retrieved for features that share a common substring. An illustration is presented in Figure 1. In the example, a basic assumption in the calculation of melting temperature, for a feature that is segmented by one or more internal nodes, is that the formula for the calculation is additive. The formulation will not be discussed here as various methods of calculation are available, including the Wallace method [10], the Nearest Neighbour method and its variants [11] as well as the GC% method [12]; each having their own set of criteria and constraints. However, this assumption may not always be true and may sometimes require manipulations of existing formulas to fit the requirements of the ES-Tree. This assumption is also imposed onto other properties that are stored in the nodes.

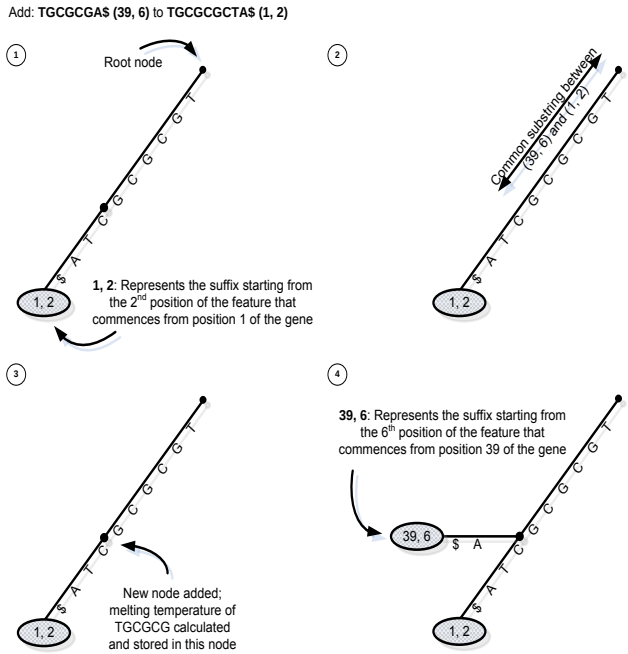

Figure 1: EF-Tree Construction

Besides storing important information at each node, the EF-Tree also facilitates the search for features with repeating substrings in combination with features that are 5' or 3' biased. As such, the EF-Tree presents a solution that aids in the selection of candidate probes that meet a combination of criteria.

There are many inherent properties in an EF-Tree that can be harnessed in identifying DNA signatures, such as the recognition of repeating substrings, common or longest common substrings and exact string matching, among others. Coupling these applications with the properties of an EF-Tree, various genomic applications can be conjured. In the followings, we present some applications that can either be used individually or in combination to satisfy various criteria that can be set.

2.2 Exploitation of Features with Repeats

In an EF-Tree, a repeat is indicated by the presence of an internal node which dictates that the sequence (path) from the root to the node is shared by the leaves resulting from the internal node. Instead of considering only the maximal repeats, all possible repeats may be considered from the EF-Tree. This is because other factors may be considered in, for example, the probe selection process. Inherently, repeats are either maximal repeats or suffixes of the maximal repeats and are hence biased towards the 3' end of the feature.

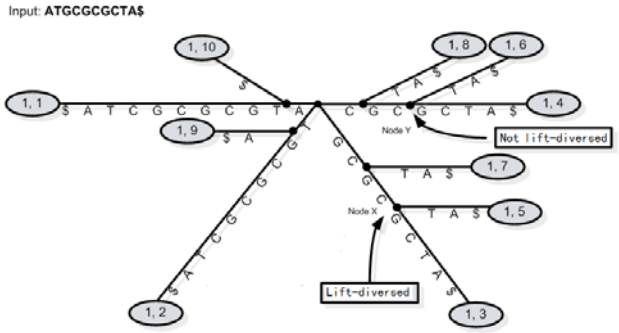

Figure 2: An EF-Tree with GC repeats

As illustrated in Figure 2, a maximal repeat is identified at node X because it is lift-diverse while a non-maximal repeat is identified at node Y because it is not lift-diverse. Lift-diversity is used to determine if the substring represented by an internal node is a maximal repeat [9]. While it may be desirable to utilize the sequence represented by node X – so as to ensure that the repeat bound is specific, it may not always be appropriate because of factors like melting temperature and GC content. Hydrogen bonding between GC is known to have higher enthalpy because there are 3 hydrogen bonds as opposed to AT, which has only 2. As such, given the extra energy required to break the bonds, the sequence represented by node X may be selected against because it may not meet the criteria set for the melting temperature.

On the other hand, if we are to consider all repeats instead of just the maximal ones, then the sequence represented by node Y, which is not a maximal repeat but a substring of node X, may be considered for selection as a candidate signature. By not limiting the search of repeats to only maximal ones, suitable features may be selected as candidate signatures.

In addition, because the repeats in an EF-Tree represents not only a single feature, but a set of features, an advantage of using an EF-Tree is the ability to identify repeats across different features vis-à-vis an internal repeat within a single feature which is represented by a tree for a single feature or a signature. Features that contain the repeat represented by the node are indicated at the leaves of the subtree of that node.

2.3 Acquisition of Variable Length Features

The generation of features depends on the length of the sliding window, which will determine the length of the features as well as the number of features available for selection. Ultimately this is also dependent of the size of the candidate signatures that is desired. While it may seem reasonable to take the length of the probes as the length of the sliding window, additional considerations may be required in the use of an EF-Tree.

If the selection of repeats or maximal repeats is used as a feature selection strategy, then the use of just the repeating sequence as a feature, or the use of the entire feature represented by the repeating sequences, will determine the length of the sliding window. If the former is used, then a longer window length may facilitate the selection of longer repeats, if such repeats exist. An illustration is shown in Figure 3 where *a* represents a repeat or a common substring between (1, 3), (1, 4), (1, 5) and (1, 7), while a concatenation of *a* and *b* represents that of (1, 3), (1, 5) and (1, 7) and a concatenation of *a*, *b*, and *c* represents that (1, 5) and (1, 7). If a longer window length is used, then longer repeats may be obtained, if they exist. Moreover, longer probes may sometimes be desired because it is considered to provide a higher level of sensitivity [13]. However, the use of any of the 3 possible repeats may be further constrained by the melting temperature or GC content.

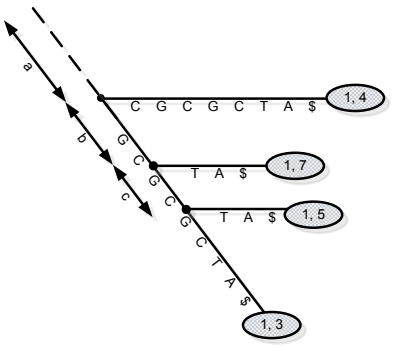

Figure 3: Segmentation of features

On the other hand, a sequence may not have repeats, and thus the use of such a strategy should be predetermined by some form of domain knowledge with respect to the genome. In this case, the entire

feature represented by the repeats may be used, and it may not be necessary to increase the length of the sliding window by more than what is dictated by the desired probe length. Therefore, the availability of such an application is not a means to an end, but rather to facilitate the overall probe selection strategy.

2.4 Elicitation of Biased Features

Selection of features that are biased towards either end of the gene may sometimes be desired if there is domain knowledge that indicates the uniqueness of target sequences at one end. This could be done by selecting leaves that represent features with a lower feature or suffix start position number – for features biased to the 5' end, or a higher number, for features biased to the 3' end. An example is shown in Figure 4: two leaves in the format (*a, b*), where *a* represents the start position of the feature within the target sequence, while *b* represents the start position of the suffix within the feature. From these leaves, it is possible to streamline, or prune, trees such that only suffixes representing a higher value of *a* – for 5' biased features, or a lower value of *a* – for 3' biased features, remains. The cut-off value may be determined arbitrarily or by using a value that cuts the number of features by half. This effectively reduces the problem domain by leaving out features, or suffixes of features.

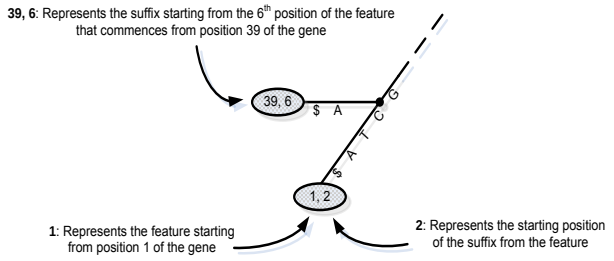

Figure 4: Leaves of an EF-Tree

Sometimes, a single leaf may represent more than one feature, as shown in Figure 5. These leaves represent suffixes of the same length, and hence the same start position from different features. In Figure 5, a leaf is shown to represent suffixes from a feature that starts from position 1 of the target sequence as well as one that starts from position 45, hence representing a common substring – or a repeat sequence, between the two suffixes and their respective features. Given a hypothetical signature of length 100, and a sliding window of 50, the last feature starts from position 51 of the signature, (1, 2) is thus 5' biased and (45, 2) is 3' biased. In pruning the EF-Tree, the decision to remove this leaf will depend on the signature selection strategy. The advantage of using an EF-Tree is the ability to provide contextual information, relative to the entire set of features that are available for signature selection in a single tree.

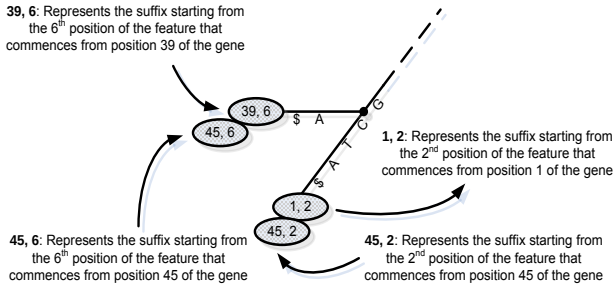

Figure 5: Leaves representing more than one feature

3 Hybrid Computing with FPGA

3.1 Analysis on Space and Time Complexity

The node of an EF-Tree is represented together with the incoming edge label information. Each node contains two integers representing the start and end position of the corresponding substring. As the end point can be deduced from the start position of the internal node (child) or is simply n (for a leaf node), it is sufficient to store the start position of the substring.

Each node has pointers to all its internal nodes and they can be represented as an array, as a linked list or as a hash table. If the size of the alphabet $\sum$ is small, then an array of size $|\sum|$ can be used to represent the child node pointers. The child node whose incoming label starts with the *ith* character in a ranked alphabet is represented by the *ith* entry in the array. Consider the tree space for inputs where *n* is less than the largest 4-byte integer, that is, $\log n < 32$. For a string of length *n*, the tree has 1 root, *n* leaves, at most *n*-1 internal nodes, and at most 2*n*-2 edges.

For each leaf node, we may store a pointer to its parent, and store the starting index of the suffix represented by the leaf, for 2*n* words of storage. Storage for each internal node may consist of 4 pointers, one each for parent, leftmost child, right sibling and suffix link, respectively. This will require approximately 4*n* words of storage. Each edge label consists of a pair of integers, for a total of at most 4n words of storage. Putting this all together, the implementation of EF-Tree takes 10*n* words or 40*n* bytes of storage.

Theoretically, the EF-Tree can be constructed in linear time and requires linear storage; that is, for a string *S* of length *n*, the tree can be built with the

time complexity of O(*n*), if the letters come from an alphabet of integers in a polynomial range. For larger alphabets, the running time is dominated by first sorting the letters to bring them into a range of size O(*n*), and this is with the time complexity of O(*n* log *n*).

3.2 Enhanced Suffix Array

In practice, an EF-Tree can be implemented as a suffix array [14, 15] which consists of the indices of the first characters of each suffixes in the sequence analyzed. Each suffix can be completely specified by the index of its first character in lexicographical order in the text. For example, given a sequence string:

T [0, 6] = $\overset{1}{A}\overset{2}{T}\overset{3}{C}\overset{4}{G}\overset{5}{C}\overset{6}{G}\overset{7}{G}$

The suffix array in lexicographical order is as follows:

$\overset{1}{A}\overset{2}{T}\overset{3}{C}\overset{4}{G}\overset{5}{C}\overset{6}{G}\overset{7}{G}$
$\overset{3}{C}\overset{4}{G}\overset{5}{C}\overset{6}{G}\overset{7}{G}$
$\overset{5}{C}\overset{6}{G}\overset{7}{G}$
$\overset{4}{G}$
$\overset{4}{G}\overset{5}{C}\overset{6}{G}\overset{7}{G}$
$\overset{6}{G}\overset{7}{G}$
$\overset{2}{T}\overset{3}{C}\overset{4}{G}\overset{5}{C}\overset{6}{G}\overset{7}{G}$

The suffix array, SA = (1,3,5,7,4,6,2)

The naïve suffix array is the lexicographically ordered leaf list of the EF-Tree. However, the array misses significant information due to the absence of internal nodes. An enhanced suffix array can be introduced to compensate the missed information.

Table 1: Representation of enhanced suffix array

i	suf	lcptab	suffix
0	1	0	ATCGCGG
1	3	0	CGCGG
2	5	2	CGG
3	7	0	G
4	4	1	GCGG
5	6	1	GG
6	2	0	TCGCGG

We add two data structures, an LCP (longest common prefix) table and the LCP interval table, into the array. In the LCP table, lcptab[i] stores the length of longest common prefix of the suffixes, suf[i] and suf[i-1]. The LCP interval table, lcpinterval[i,j] stores the length of longest common prefix of the suffixes suf[i] and suf[j]. Computation for feature identification in the following FPGA implemented algorithm can be reduced because lcptab[i,j] equals the length of the path label of the lowest common ancestor of the two leaves representing suf[i] and suf[j].

Thus, for the above string, ATCGCGG, we have the representation as in Table 1.

3.3 Mapping and Routing on FPGA

Hybrid computing is the strategy of deploying multiple types of processing elements within a single workflow, and allowing each to perform the tasks to which it is best suited. Complementing general-purpose microprocessors with specialized processing elements, we deployed an architecture with reconfigurable computing fabrics, on-chip parallel processing elements designed for compute-intensive applications with private, and software-controlled local memories [16].

For effective running of the DNA signature identification routines, we eventually map the codes for the suffix array implementation on a RCHTX Virtex-4 FPGA card. The computing system is the Tyan/D 1207/V/2GBL/PCIE server compliant with the HyperTransport (HTX) interface standard. The HTX socket provides high-bandwidth communication between the Xilinx Virtex-4 FPGA co-processor and AMD's Opteron processors. Such a hybrid computing system (see Figure 6) provides power-efficient acceleration technology that increases system performance, scalability and flexibility [17].

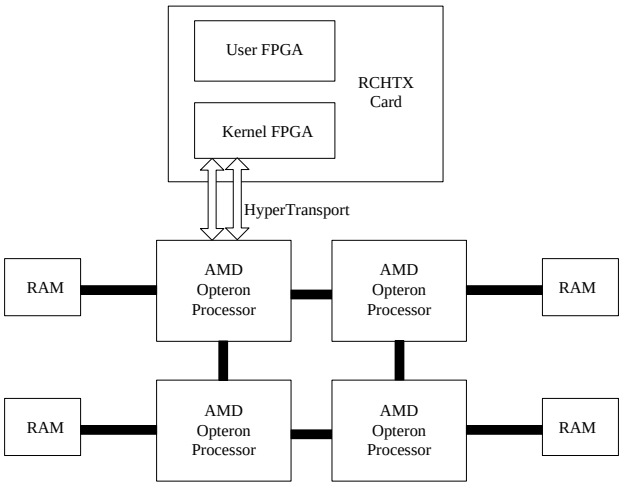

Figure 6: HTX-compliant Hybrid System

In the system, application interface and sequence data access functions are implemented on the AMD Opteron processors, while the core suffix array

construction algorithm is mapped on the User FPGA of the RCHTX card. The working data is transferred between the host CPU and FPGA via the HyperTransport interconnect.

Codes are developed with Handel-C, compiled into the low-level Hardware Description Language (HDL), and finally routed as images of FPGA for parallelism. The core algorithm is described as follows:

1. Convert the DNA string into DNA binary sequences in their respective binary array: seqA[n], seqC[n], seqG[n], seqT[n], where n is the length of the DNA sequence.
2. Specify initial data of variables and arrays, where n is the length of the DNA sequence:
   a. Initialize 5 binary bins: binE[n], binA[n], binC[n], binG[n], binT[n] to be equal to seqs. bins record the ith character for each suffix, where the ith value is the number of next iteration. binX[j] = 1 in (i-1)th iteration indicates that the ith character of the jth suffix in lexicographical till now is X, where E refers to empty.
   b. Initialize group variables: st[n] = 1, end[n] = n, plus[n] = 0.
   c. Initialize iteration number to be 0.
   d. Initialize suffix array, suffix[i] = i+1, for i is from 0 to n-1.
3. For i = 0 to n-1, calculate the number of positions of each character in the current group where the suffix will be allocated.
   sumX[i] = $\sum_{x=start}^{end} binX[x]$, where X is the character (E,A,C,G,T), start and end are the start and end position of the group where the suffix will be allocated.
4. For i = 0 to n-1, calculate the position of suffix in the cluster of current group where the suffix will be calculated; reset the finish variable to be binary 1.
   hex[i] = $\sum_{x=start}^{i} binX[x]$, where X is the character such that binX[i] == 1.
5. Calculate the order of the suffix based on ith character in lexicographical order. For the character that binX[i] == 1
   pos[i] = hex[i] + plus[i] + $\sum_{Y=E}^{Y<X} sumY[i]$
   where hex[i] indicates the position in the current cluster, plus[i] is the position before current group, the third addend is the sum of smaller characters in the current group.
6. Re-arrange the order of the suffix and the 5 bins:
   suffix[pos[i]-1] = suffix[i], binX[pos[i]-1] = binX[i]
7. Calculate the clustering grouping for the ith character of the suffix. For i = 0 to n-1, if binX[i] == 1,
   st[i] = st[i] + $\sum_{Y=E}^{Y<X} sumY[i]$
   end[i] = st[i] + $\sum_{Y=E}^{Y<=X} sumY[i]$ - 1
   plus[i] = plus[i] + $\sum_{Y=E}^{Y<X} sumY[i]$
8. Set the 5 bins to the next character in each of the suffix
   For i = 0 to n-1
   if (suffix[i] + iteration > n-1)
   binE[i] = 1, binX[i] = 0;
   if (suffix[i] + iteration < n)
   binE[i] = 0, binX[i] = seqX[suffix[i] + iteration]; where X is A,C,G,T ;
   if (hex[i] > 1)
   set finish = 0;
   iteration ++
9. Repeat steps 3,4,5,6,7,8 if iteration < n and finish is 0.

Following the example in Section 3.2, we show the FPGA mapping and routing of the data structures and processes as follows:

1. DNA[n]: ATCGCGG

2. 1-bit arrays: seqA[n], seqC[n], seqG[n], seqT[n], where seqX[i] = 1 means the ith position of the DNA sequence is character X, as in Table 2.

Table 2: 1-bit array seqX in FPGA implementation

bin\suffix	1	2	3	4	5	6	7
seqE	0	0	0	0	0	0	0
seqA	1	0	0	0	0	0	0
seqC	0	0	1	0	1	0	0
seqG	0	0	0	1	0	1	1
seqT	0	1	0	0	0	0	0

3. 1-bit arrays: binA[n], binC[n], binG[n], binT[n], to record the ith character for each suffix, where the ith value is the number of next iteration. After first iteration, the second character for each suffix is

TGGCGEC, thus the values filled are shown in the Table 3.

Table 3: 1-bit array binX in FPGA implementation

bin\suffix	1	3	5	4	6	7	2
binE	0	0	0	0	0	1	0
binA	0	0	0	0	0	0	0
binC	0	0	0	1	0	0	1
binG	0	1	1	0	1	0	0
binT	1	0	0	0	0	0	0

4. st[n] , end[n], plus[n]

Clustering groups for the ith character of the suffix, that is, in iteration i, they contain clustering group information of the ith number. For example, after sorted in iteration 1, the first character of each suffix is: A,C,C,G,G,G,T, thus

st = (1,2,2,4,4,4,7)
end = (1,3,3,6,6,6,7)
plus = (0,1,1,3,3,3,6)

5. sum[E] sum[A] sum[C] sum[G] sum[T]. They determine the number of positions of each character in the current group where the suffix will be allocated. For example, for iteration 2, the second character of each suffix is: T,G,G,C,G,E,C, thus the values filled are shown in Table 4.

Table 4: Array sumX in FPGA implementation

sum\suffix	1	3	5	7	4	6	2
sumE	0	0	0	1	1	1	0
sumA	0	0	0	0	0	0	0
sumC	0	0	0	1	1	1	1
sumG	0	2	2	1	1	1	0
sumT	1	0	0	0	0	0	0

6. suffix[n] - Suffix array after iteration i.

7. hex[n] - The position of suffix in the group of the cluster to be allocated. For example, in the first iteration, the first character for each suffix is: A,T,C,G,C,G,G, thus, hex = (1,1,1,1,2,3,4). In the second iteration, the second character for each suffix is: T,G,G,C,G,E,C, thus, hex = (1,1,2,1,1,1,1). In the third iteration: the second character for each suffix is: C,C,G,E,G,E,G, thus hex = (1,1,1,1,1,1,1).

4 Identification of DNA Signatures for Influenza A Virus

In the experiment, we aim to find a set of DNA signatures for influenza A virus that could be used in disease diagnostic applications. Influenza A virus genome sequences are used as test sequences in the development of our pipeline because they exist in a multitude of closely related sequence variants for which many complete genomic sequences of each subtype are available in the public databases. The size of the influenza A virus genome is relatively small, approximately 8kb, enabling quick yet rigorous proof of principle testing.

From the 1st cycle of the workflow of finding common substrings between the first 2 virus subtypes, H1N1 and H3N2, there were 2429 LSCs. These LSCs were concatenated into a "reference sequence" with a spacer sequence ("PPPPP") in between each LSC. Results of common substring determination at increasing pipeline iterations are shown in Table 5.

Table 5: Common substring determination

Iteration	Inputs	No. of unfiltered common substrings	No. of clusters obtained by DNACLUST (= no. of unfiltered longest common substrings per cluster identified)	No. of longest common substring per cluster (LSC) of size 20bp and above identified
1	H1N1 and H3N2	8963	2429	104
2	H1N2	7582	2221	84
3	H9N2	5616	2280	23
4	H5N1	4528	2005	20
5	H7N7	3883	1724	15
6	H7N2	3476	1526	12
7	H7N3	3086	1346	13

After the final 7th iteration of the pipeline, a total of 13 LSCs varying in size from 20 – 50bp were identified. These 13 LSCs therefore represent 13 DNA regions containing sequences≥ 20bp that are common to all of the 8 original input influenza genome sequences. In other words, these represent the DNA signatures that can be used in the design of specific probes for use in biological applications.

To check the specificity of the DNA signatures identified by our pipeline, i.e. to verify if probes designed using these DNA signatures are unique to Influenza A virus only, and if this specificity extends to only particular subtypes, we performed BLAST analysis against GenBank 'nr' database using the NCBI BLAST web server (http://blast.ncbi.nlm.nih.gov/Blast.cgi). We used the program BLASTN with settings Word size =15. We first performed BLAST analysis on each of the 13 DNA signatures against the original 8 sequences to verify that they are the common substrings of the 8 sequences. The results showed that the 13 DNA signatures were correct for the tested 8 Influenza virus A subtypes.

Next we checked if the DNA signatures were specific to Influenza A virus by performing BLAST analysis using each of the 13 candidate probes as query sequences. The results showed that the substrings were highly specific to only Influenza A virus, and not to the B or C genera. Each probe aligned to a unique region in the virus genome, with a BLAST maximum identity of 100% across each entire query length. No query aligned to more than 1 region in the genome, confirming high sequence specificity and hence suitability for use as probes.

The longest DNA signature, signature 1 (50bp), aligned fully to a region in segment 7. There were four other DNA signatures (signatures 3, 4, 9 and 12) that also aligned to various regions of segment 7. There were two DNA signatures (signatures 2 and 13) which aligned to various regions in segment 1, four DNA signatures (signatures 7, 8, 10 and 11) which aligned to different regions in segment 3, one DNA signature (signature 5) which aligned to a region in segment 5, and one DNA signature (signature 6) which aligned to a region in segment 2. Interestingly, no candidate probes were identified in this experiment that mapped to segments 4, 6 or 8.

To determine and display the positions of the regions on Influenza A virus which the DNA signatures mapped to, we used an arbitrary "reference sequence" as one of the test genome sequences, H1N1 (accession numbers NC_002016.1 to NC_002023.1). The start and end positions of each probe aligned to this reference sequence show the alignment of each probe.

The E-values from our BLAST results ranged from 0.92 to 1.00e-18. In general, the lower the E-value, or the closer it is to zero, the more "significant" the match is. However, despite the 20bp probe having a high E-value (E = 0.92) due primarily to E-value calculations being influenced by query length, the actual BLAST alignments clearly show that it is still highly specific, mapping only to Influenza A virus sequence like its longer counterparts.

We further examined the subtype specificity of the DNA signatures, by counting the number of times each signature aligns to a different virus subtype. We performed BLAST analysis and select the maximum number of aligned sequences as "1000". Results show that the subtype specificity distribution pattern by the DNA signatures is similar to the Genbank database subtype-specific distribution, except for H9N2 and H5N1. Both subtypes are under-represented in the distribution, only 1.5% and 4.5% of the BLAST-matches by the DNA signatures are specific to H9N2 and H5N1, respectively.

5 Discussion and Conclusion

With the free availability of thousands of complete genome sequence assemblies such as the GOLD database at http://www.genomesonline.org/cgi-bin/GOLD/bin/gold.cgi, we are conducting more experiments on identification of DNA signatures in genomic sequences.

We would like to use our high-performance whole genome algorithm to identify signature sequences in bacterial and viral genome sequences, and to use these unique sequences as a basis for diagnostic assays to detect and genotype microbes in both environmental and clinical samples. For examples, *Bacillus anthracis*, *Bacillus cereus* and *Bacillus thuringiensis* are genetically so close that it has been proposed to consider them as a single species. However, these bacteria are very different on a phenotypic level. *B. cereus* is a food contaminant, *B. thuringiensis* is a useful bacterium used as a pesticide, while *B.anthracis* is a virulent pathogen for mammals, and has been used as a bioterror and biological warfare agent. In another example, *Brucella melitensis* and *B. suis* have adapted to different hosts (cattle and swine, respectively) and yet the genomes of the two *Brucella* species differ by only 74 open reading frames out of 3378 open reading frames.

In a more comprehensive experiment, the success of the proposed approach critically depends on the features used to identify signatures that can, in turn, accurately differentiate between target genomes and sample background. With the DNA sequence information, in addition to identifying signatures, the products of the genes of interest can also be readily produced and studied; furthermore, mutants can be prepared for genetic and functional analysis.

The pipeline currently works well for genome sizes (input strings) of a few thousand basepairs. However, we have found that for sequences 1Mb and longer, we encounter memory limitations because suffix trees are extremely memory intensive by nature. The implication of this is that the pipeline is currently well suited to identifying DNA signatures in virus sequences, but for bacteria, fungi or even more complex organisms, a workaround will have to be developed to make the process more memory efficient.

Apart from the exact string matching problem, there can be other sequence-based problems such as approximate matching, database searching and so on. For instance, the design of probes for promoter regions is an example of approximate matching. As promoter region are often consensus inferred from a set of upstream sequences, exact matches are seldom observed. A suffix tree is a versatile data

structure that can be used to solve a variety of sequence-based problems, including exact / approximate matching, database querying, genome alignment and so on. The existing suffix-tree based pipeline may be further developed into a more versatile format in future, perhaps incorporating approximate matching.

In conclusion, DNA sequences that are unique to a given species or strain, or to a defined group of related organisms, can be used to distinguish the target from unrelated species. Identification of such DNA signatures is especially important for detecting organisms that may be pathogenic to humans. In this project, we have used a compressed suffix-tree based algorithm in the pipeline to identify multiple DNA signatures common to a set of input sequences. Using the genome sequences of 8 Influenza virus A subtypes known to infect humans, our pipeline was able to correctly identify 13 DNA signatures from 20 – 50bp in size, that should be useful as biosignatures or candidate probes. We examined the specificity of these candidate probes and confirmed that they are highly specific to only Influenza A virus, but are not host-specific.

Acknowlodgment:
This work is partially supported by a grant (M4080106.020) by Nanyang Technological University and another (M4080634.B40) by NTU Institute for Media Innovation.

References:

[1] Metzker, M.L., Sequencing technologies - the next generation. *Nat Rev Genet*, 2010. Vol. 11, No.1, pp31-46

[2] Teufel, A., et al., Current bioinformatics tools in genomic biomedical research (Review). *Int J Mol Med*, 2006, Vol. 17, No. 6, pp967-973

[3] Smith, T.F. and Waterman, M.S., Identification of common molecular subsequences. *J Mol Biol*, 1981, Vol. 147, No. 1, pp195-197

[4] Needleman, S.B. and C.D. Wunsch, A general method applicable to the search for similarities in the amino acid sequence of two proteins. *J Mol Biol*, 1970, Vol. 48, No. 3, pp443-453

[5] Ko, P. and Aluru, S., Suffix Trees Application in Computational Biology, in *Handbook of Computational Molecular Biology*, S. Aluru, Editor. 2006, CRC Press: Boca Raton, FL, USA. pp166-193

[6] Weiner, P., Linear pattern matching algorithms, in *Proceedings of the 14th Annual Symposium on Switching and Automata Theory (swat 1973)*, IEEE Computer Society, 1973

[7] McCreight, E.M., A Space-Economical Suffix Tree Construction Algorithm. *Journal of the ACM*, 1976, Vol. 23, pp262-272

[8] Ukkonen, E., On-line Construction of Suffix Trees. *Algorithmica*, 1995, Vol. 14, pp249-260

[9] Gusfield, D., Algorithms on Strings, Trees and Sequences, *Cambridge University Press*, New York, 1997

[10] Wallace, R.B., et al., Hybridization of synthetic oligodeoxyribonucleotides to phi chi 174 DNA: the effect of single base pair mismatch, *Nucleic Acids Research*, 1979. Vol. 6, No. 11, pp3543

[11] SantaLucia, J.J., et al., Improved Nearest-Neighbour Parameters for Predicting DNA Duplex Stability, *Biochemistry*, 1996, Vol. 35, pp3555-3562

[12] Flikka, K., et al., XHM: A System for Detection of Potential Cross Hybridizations in DNA Microarray, *BMC Bioinformatics*, 2004, Vol. 27, No. 5, pp117

[13] Simmler, H. and Singpiel, H., Real-Time Primer Design for DNA Chips, *2nd IEEE International Workshop on High Performance Computational Biology*, Nice, France, 2003

[14] Abouelhoda, M.I., Kurtz, S., Ohlebusch, E., Replacing suffix trees with enhanced suffix arrays, *Journal of Discrete Algorithms*, 2004, Vol. 2, No. 1, pp53-86

[15] Giegerich, R., Kurtz, S., Stoye, J., Efficient implementation of lazy suffix trees, Software-Practice and Experience, 2003, Vol. 33, No. 11, pp1035-1049

[16] Stepanova, M., Lin, F. and Lin, L.C.V., In silico modelling of hormone response elements, *BMC Bioinformatics*, 2006, Vol. 7, No. 4

[17] Stepanova, M., Lin, F. and Lin, L.C.V, A Hopfield Neural Classifier and Its FPGA Implementation for Identification of Symmetrically Structured DNA Motifs, *The Journal of VLSI Signal Processing*, 2007, Vol. 48, No. 3, pp239-254

[18] Ng, K. L., Huang, C. H. and Tsai. M. C, Vertebrate microRNA genes and CpG-islands, *WSEAS Trans on Biology and Biomedicine*, Vol. 7, Issue 3, July 2010, pp73-81

[19] Yamada, Y. and Satou, K., Prediction of Genomic Methylation Status on CpG Islands Using DNA Sequence Features, *WSEAS Trans on Biology and Biomedicine*, Vol. 5, Issue 7, 2008, pp153-162

[20] Qi, Y., Lin, F. and Wong, K. K., High Performance Computing in Protein Secondary Structure Prediction, *WSEAS Trans on Circuits and Systems*, Vol. 2, No. 3, 2003, pp619-624