This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCBB.2018.2885296, IEEE/ACM Transactions on Computational Biology and Bioinformatics

Hardware-Software Codesign based Accelerated and Reconfigurable Methodology for String Matching in Computational Bioinformatics Applications

Venkateshwarlu Y. Gudur, and Amit Acharyya, *Member, IEEE*

Abstract— Research for new technologies and methods in computational bioinformatics has resulted in many folds biological data generation. To cope up with the ever increasing growth of biological data, there is a need for accelerated solutions in various domains of computational bioinformatics. In these domains, string matching is a most versatile operation performed at various stages of the computational pipeline. For search patterns that are updated with time, there is a need for accelerated and reconfigurable string matching to perform faster searching in the ever-growing biological databases. In this paper, we have proposed an accelerated and real-time reconfigurable methodology for string matching using hardware-software codesign. Using state of the art field programmable gate arrays we have proposed a complete system-on-chip solution for applications that require accelerated as well as real-time reconfigurable string matching. The proposed methodology is the first of its kind novel approach for high-speed string matching that also supports quick reconfiguration by patterns changing with time. It is verified at the string matching stage of protein identification. Experimental results show that the architectures designed using our proposed methodology are 4X faster than state-of-the-art software implementation running on a workstation and 1.5X-4X faster than hardware accelerators available in the literature.

Index Terms— Hardware acceleration, hardware-software codesign, real-time reconfiguration, reconfigurable SoC FPGA, string matching

1 Introduction

A large amount of genomics data is contributed to the life science society due to high-throughput next generation sequencing methods and it is doubling at every 18 months [5]. Furthermore, due to the advancements in liquid chromatography and mass spectrometry, mass spectra data in proteomics study is getting generated at a very high rate [29]. As a result of the increasing growth of data in genomics and proteomics, operations involved in various applications of bioinformatics have become significantly compute intensive. String matching [1] is one such operation that is most widely used in the field of computational and information systems with many real-world applications including web search engines, information retrieval, intrusion detection, pattern recognition and finding locations of nucleotides and amino acids in the field of computational bioinformatics [6], [12], [21], [22]. Aho-Corasick (AC) algorithm is a widely used string matching algorithm in bioinformatics with various applications including sequence alignment, locating nucleotides, proteogenomic mapping, etc. [6], [8], [15], [19].

In applications of bioinformatics including biomarker discovery, basic local alignment search tool, homologous series detection, etc., patterns to be searched are constantly varying and these demands to configure the underlying systems with latest patterns [8], [22], [29]. Real-time reconfigurable string matching is required in these applications to search patterns that are continuously updated with time. Since the aforementioned applications are computationally intensive due to the data growth [5], high speed methodologies are of utmost importance to perform reconfigurable string matching. In this context software, high-performance computing and hardware acceleration are the methodologies reported in the literature [3], [24]. Although software methods are flexible and reconfigurable, their running time for string matching is turning to be a limiting factor due to the growing size of biological databases [3], [4], [13], [16]. On the other hand, high-performance computing solutions involve a large computational infrastructure including multiprocessors, multicore CPUs, clusters, cloud and grid computing, etc., making their management very costly [24]. Hence researchers emphasized on hardware accelerators employing cost-effective field programmable gate arrays (FPGAs).

Computationally intensive algorithms of bioinformatics are efficiently implemented and accelerated using high-density FPGAs [3], [4], [7], [14]. Numerous domains including pairwise sequence alignment, multiple sequence alignment, resequencing, gene-sequence analysis, DNA sequencing, database searching, genome assembly and study of homologous sequences are reported to use FPGAs for acceleration [3], [4]. In [7] performance of long read mapping is improved by hardware and it is imple-

• *V.Y. Gudur and A. Acharyya are with Department of Electrical Engineering, Indian Institute of Technology Hyderabad, 502285, Telangana, India. E-mail: {ee15resch02009, amit_acharyya}@iith.ac.in.*

xxxx-xxxx/0x/$xx.00 © 200x IEEE Published by the IEEE Computer Society

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCBB.2018.2885296, IEEE/ACM Transactions on Computational Biology and Bioinformatics

mented by a novel FPGA based system. A hybrid overlap searching algorithm of DNA fragments suitable for parallel implementation of FPGA is proposed in [14]. Here the time gap required for inserting a new nucleotide is utilized to compare overlapping DNA fragments and thus assist the assembly process in real-time [14].

In genomics and proteomics, often there is a requirement to complete the string matching operations in a stipulated time. These include searching genome databases to avoid data pile up and mass spectrometry based proteomics study where database search parameters are controlled for searching of peptide tags [6], [19], [34], [35], [36]. Similarly, online data analysis is part of hypotheses-driven tandem mass spectrometry (hdMS/MS). HdMS/MS require to perform data analysis under a strict real-time limits [35], [36]. In LC-MS/MS based shotgun proteomics, precursor ion selection is an important task to reduce the number of MS/MS peaks. Peptide sequences are used to avoid redundant computations and searches [36]. In all the above mentioned examples, real-time string matching is of utmost importance.

In the implementation of FPGA based hardware accelerators, various design-steps are executed. These steps include writing programs in hardware description languages (HDL), synthesis, translation, mapping, placement and routing, programming file generation and configuring FPGA using the configuration file [18], [20]. For practical designs, commercial tools require hundreds of seconds for executing synthesis, implementation and configuration file generation [16]. This methodology is unsuitable and is a limitation for applications where patterns are generated at a higher rate than the reconfiguration time of FPGAs, for example, the string matching stage of protein identification using high-throughput mass spectrometry, hdMS/MS, precursor ion selection, etc. [2], [35], [36]. In addition, to reconfigure FPGAs using this methodology, there is a requirement of a dedicated computer system and technical expertise to handle sophisticated proprietary tools.

Motivated by the aforementioned facts and limitations, it can be inferred that there is a need for accelerated methodologies for string matching that also support real-time reconfiguration. In our preliminary work, we identified this problem and addressed it partially by introducing the codesign based idea for accelerated string matching [11]. In this paper, we extend this work and propose a novel hardware-software codesign based accelerated and real-time reconfigurable methodology for string matching in computational bioinformatics applications where hardware caters the need of acceleration while software generates the reconfiguration data and also assists the hardware in real-time reconfiguration. The proposed methodology is useful for various applications that require accelerated and real-time reconfigurable string matching. The methodology is employed to accelerate and implement real-time reconfigurable Aho-Corasick algorithm for string matching in computational bioinformatics using hardware-software codesign on SoC FPGA. The proposed methodology is evaluated for the time required for reconfiguration at the string matching stage of protein identification using high-throughput mass spectrometry and it is observed that the string matching system is reconfigurable with new patterns in the order of milliseconds. To the best of our knowledge, this is the first of its kind novel work where the time required for reconfiguration of a hardware assisted string matching system for bioinformatics applications is reported in the literature. The design implemented using the XC7Z020 FPGA device running at 100 MHz is reconfigurable in the order of milliseconds and achieves 4-fold speed gain in comparison with an equivalent software implementation running on a 2.60 GHz Xeon workstation with 8 GB of memory and 1.5X-4X times faster than a hardware implementation available in the literature [8].

The novelties in this paper are summarized as follows:

- A first of its kind novel approach for high-speed string matching that also supports quick reconfiguration by patterns changing with time is proposed in this paper. The patterns that are to be matched in the database are updated to the hardware system in real-time and matching in the database is performed in an accelerated mode using hardware-software codesign.
- Two architecture design methodologies- single core and multiple core, supporting both accelerated as well as real-time reconfigurable string matching are proposed based on an intelligent partitioning and optimized designing of hardware-software codesign. For string matching in multiple large databases with a large number of patterns, the multiple core architecture is proposed that is designed using a parallel connection of memory saving single cores.

The rest of the paper is organized as follows: In section 2, we review the background and related work. The proposed methodologies for accelerated and reconfigurable string matching are presented in Section 3. Section 4 describes the experimental platform. The performance of the proposed methodologies is evaluated in section 5. Section 6 concludes the paper.

2 Related Work and Background

In this section, we present related work on string matching and its applications in computational bioinformatics. Later we present relevant background information on the hardware-software codesign.

2.1 String Matching and its Applications in Computational Bioinformatics

In string matching a database or text is searched to identify locations of strings or patterns [1], [21]. It is utilized in various disciplines of computational bioinformatics including biomarker discovery, basic local alignment search tool, proteogenomic mapping, homologous series detection, sequence alignment and sequence similarity [12], [15], [22]. These disciplines require finding locations of multiple patterns which are made of nucleotides or amino acids in databases. This requirement necessitates the use of multiple pattern matching algorithms.

Aho-Corasick (AC) algorithm, one of these algorithms, is a widely used multi-pattern string matching algorithm

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCBB.2018.2885296, IEEE/ACM Transactions on Computational Biology and Bioinformatics

[1]. Substantial research works are found on the applications of Aho-Corasick algorithm (ACA) in computational bioinformatics [2], [6], [8], [15], [19]. For fast alignment of large genomic sequences, a simplified version of ACA is presented in [6]. The SITEBLAST algorithm of [19] uses ACA for local alignment of genomic sequences using prior knowledge of biological signatures. ACA performs the best of many string matching algorithms to locate unique oligonucleotides from DNA databases [15]. In protein identification using DNA based search of [2], protein coding region within the DNA is identified and then using ACA this sequence is matched with peptides obtained using mass spectrometry. In proteogenomic mapping, where genome annotation is performed using proteomics data, peptides discovered by mass spectrometry are matched by using ACA in a genome database translated in all six reading frames [8].

2.2 Hardware-Software Codesign

Hardware-software codesign is a concurrent and coordinated design of a system that includes both hardware and software modules/components and they interact with each other to perform a complete task [25]. In this codesign based approach, the system is made up of a dedicated hardware and software that is mapped onto a central processing unit (CPU). This approach facilitates with the advantages of both hardware and software such as speed, power and parallelism of former and flexibility, modularity of later [25]. Design constraints are optimized by combining the rewards offered by both hardware and software [27]. Over the course of time, research in hardware-software codesign has evolved through a number of stages and passed through many approaches such as focusing on partition strategies, architectures, codesign approaches in multiprocessing, multithreading and multicore environments [27].

Recently, system-on-chip (SoC) that integrates all components of a computing and communication system on a single chip provides a new paradigm fo r codesign [23]. This approach uses the Zynq-7000 system on a chip FPGA family that has the capability to implement a complete hardware-software system on a single platform [23]. Such FPGAs take away the need for standalone integrated circuits for processor and hardware and provide a single chip solution. In addition to Zynq family with on chip ARM processor, there are various softcore processors that are used in hardware-software codesign. These are off-the-shelf, ready to use and synthesizable processor cores that facilitate rapid prototyping and final design implementation. They are completely customizable as per the demands of the application. OpenRISC 1200, PicoBlaze, Leon 3 and Mico32 are open-source softcore processors while MicroBlaze, PowerPC, Nios II and Xtensa LX are commercial softcore processors [9], [28].

3 Proposed Accelerated and Reconfigurable Methodology

In this section, we present our proposed methodology for string matching using hardware-software codesign. Then we give brief idea about ACA and its implementation on FPGA using memory based technique. Later we propose architectures for string matching using the proposed methodology and conclude the section with an application of the proposed methodology in computational bioinformatics.

TABLE 1
Flow of the Proposed Methodology

Algorithm 1: Algorithmic flow of the proposed methodology

Input: Sets of patterns **S**, database for searching **dbase**
Output: locations of patterns with their identification
1: **while S** is not empty **do**
2: read patterns of set **Si** and create AC finite state machine (FSM) **–sw**
3: create memory table for the corresponding FSM **–sw**
4: initiate hardware and transfer table to local memory of hardware **–sw**
5: perform string matching **–hw**
6: update results and communicate to software **–hw**
7: stay in standby till next set of patterns arrive **–hw**
8: **end**

Abbreviations: sw- software, hw- hardware.

3.1 Proposed Methodology for String Matching

In the hardware-software codesign of an application, frequently used and compute intensive functions or tasks of the underlying algorithms are offloaded to dedicated hardware while control actions are assigned to software [25], [27], [28]. Substantial speedup is achieved by employing fixed hardware accelerators for algorithmic and data processing tasks. The flexible software running on a processor core is well suited for executing control oriented and decision making tasks. From the view of all the above guidelines, we propose a novel hardware-software codesign based methodology for string matching. Based on the methodology, codesign based algorithmic flow for string matching is given in Table 1.

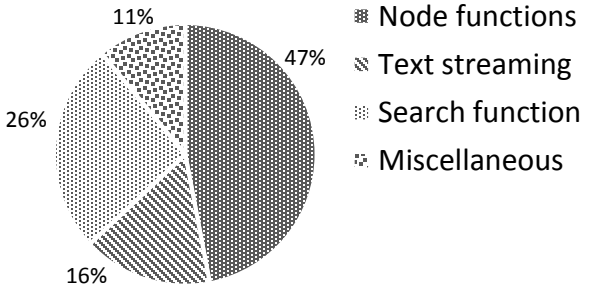

Fig. 1. Time profiling of Aho-Corasick algorithm. Values are shown as the percentage of the total time of execution of the AC algorithm.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCBB.2018.2885296, IEEE/ACM Transactions on Computational Biology and Bioinformatics

In the codesign, profiling and partitioning are performed on the desired functionality of the system [25], [27]. In software profiling, the high-level description of the system functionality represented by a programming language is analyzed thoroughly for resource-intensive and time-consuming operations and functions. A profiling software accurately analyze the program and provides useful insights of the various operations and functions carried in the program. In order to analyze the string matching using the Aho-Corasick algorithm, we have performed the time profiling operation on the algorithm implemented using C program. Profiling results obtained by running the program on multiple test cases are shown in Fig. 1. It is observed that 73% of the time, which is a significant amount, is consumed in executing the search function and operations between the nodes of AC finite state machine (FSM). Partitioning is performed to implement the node functions and search function in hardware, while data control tasks are assigned to software. In order to reconfigure the system in real time with changing patterns, the functions implemented in hardware are desired to be updated fast and efficiently. This profiling and partitioning form the core of the proposed methodology using hardware-software codesign.

For a number of sets of patterns **S**, string matching operation is required to identify these patterns in the database **dbase**. Let $\mathbf{S_i}$ represents a set of patterns that are changing with respect to time, while $\mathbf{S_{i+1}}$ represents the next set of patterns. The problem statement is to reconfigure the string matching system in real time for the patterns in $\mathbf{S_i}$ and scan the database for these patterns.

In the proposed methodology, the software reads the patterns and creates an AC finite state machine for these patterns. Using the technique of memory based FSM implementation presented in [26], the software creates a memory table for the corresponding FSM. A hardware accelerator is designed to take characters from the **dbase** and look for patterns of $\mathbf{S_i}$ in these characters. Software behaves as a master and initializes the hardware. It transfers the memory table of patterns to the hardware accelerator. String matching operation is performed by hardware and the search results are updated to the software. These steps are repetitively run for the next set of patterns of $\mathbf{S_{i+1}}$. In the design, as soon as the patterns of $\mathbf{S_i}$ are available, the system has to be reconfigured with them. This is an essential feature, for the system to be called as runtime reconfigurable.

Next, we brief about the technique of memory based FSM implementation and present the hardware architecture for the search engine. Subsequently, we introduce the hardware-software codesign based architecture for real-time string matching using the above methodology. We first present a single core architecture for accelerated and reconfigurable string matching followed by a flexible multi-core architecture for the same.

3.1.1 Introduction to Aho-Corasick algorithm with illustration

In this sub-section, we brief about ACA with an example. Its linear time complexity and the ability to simultaneously identify multiple patterns in a given text make ACA advantageous than other string matching algorithms [1], [15]. Consider four patterns AC, DAC, ABD and ACED. In ACA, the group of patterns are preprocessed to create an FSM. A large text or database is searched for these patterns by passing the characters from the text to the FSM. Fig. 2 (a) shows the FSM generated by ACA for the four patterns. In the FSM each circle is called as state or node while the initial state or root state is denoted by state 0. When a character is read from the text by the FSM, there is a transition of state from one to another for the corresponding character. These transitions are represented by edges or branches labelled with the corresponding character. Root state is retained when no part of any pattern is present in the text while other states indicate partially or fully identified patterns. A double circle is for the state where a valid pattern is found. Edges are made of normal and failure transitions. Normal transitions occur when specific characters are matched while failure transitions are useful for finding patterns that overlap with other patterns. Failure transitions to root node are not represented in the diagram for the sake of simplicity.

3.1.2 Memory based FSM Implementation on Hardware

In this sub-section, we realize the FSM generated by ACA using memory based implementation. Owing to the advances of FPGA technologies and the inclusion of on-chip memory, memory based implementation of FSMs on FPGAs are receiving considerable attention [26]. Block RAM (BRAM) memories can be read and written during operation runtime and using them different FSM circuits are implemented [26]. Besides the advantage of runtime update, for BRAM memories there is no necessity of running the time consuming synthesis, placement and routing operations as required in the case of logical elements based distributed RAM.

In a processor or computing machine, FSM can be represented in the form of a memory table [8], [26]. Fig. 2 (b) shows hardware details of the block RAM based FSM for the patterns considered in the example. A tabular representation of the FSM example is also shown alongside the hardware. String matching operation using the FSM table is performed in the following manner. At any given time the control is at one of the states in the FSM and in the table, a row corresponding to that state is read. This row of memory is connected as inputs to the multiplexer. The multiplexer selects a particular input specific to the character read from the database. We use the word cell to denote a part of the register in memory. Inputs to the multiplexer are state-cells while the cells corresponding to output match vectors are called output-cells. The control shifts to the next state as determined by the contents of the state register, which in turn holds the state-cell. The pattern match column indicates the patterns that are matched for a given state. A null value in the output-cell from pattern match column indicates that there are no matches for the given state while a non-null value indicates there is a match of patterns. For a non-null value, bits of the output-cell indicate their corresponding pat-

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCBB.2018.2885296, IEEE/ACM Transactions on Computational Biology and Bioinformatics

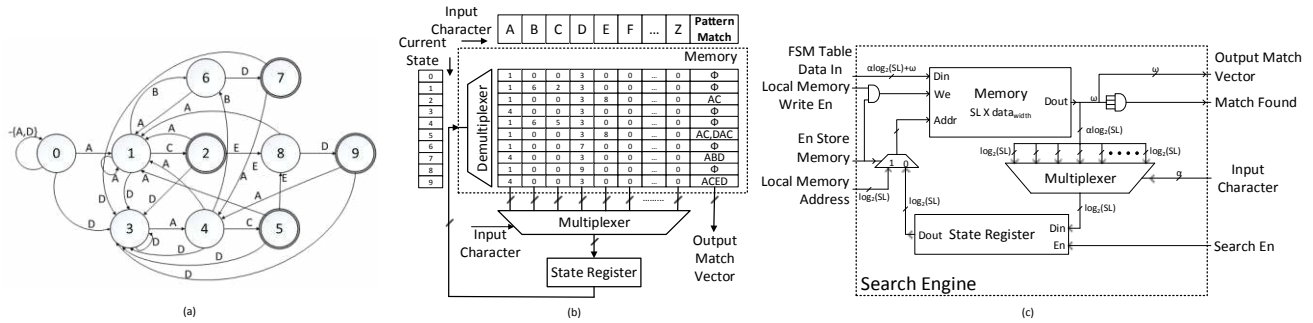

Fig. 2. Aho-Corasick algorithm: (a) FSM illustration for string matching with patterns AC, DAC, ABD and ACED. (b) Hardware details of the illustration (c) Detailed hardware architecture of the search engine

terns that are identified in the database.

3.1.3 Calculation of memory consumption of FSM

The FSM table is stored in a block RAM whose depth and width are dependent on the size and number of patterns. For S patterns each of maximum length L, the maximum number of states that are generated in the Aho-Corasick FSM is SL, which in turn decides the maximum depth of the block RAM. Each cell in a column except the cells of the pattern match column holds a state value. The width of a state-cell is the ceiling value of $\log_2(SL)$. If the width of the pattern match column is ω bits, for a given alphabet of α symbols the width of each register in the block RAM is $\alpha \log_2(SL)+\omega$ bits. The maximum number of bits required to represent a character from the alphabet is decided by $\log_2(\alpha)$. The maximum number of overlapping patterns that are identified in any state is decided by the value of ω. Based on the input character symbol, an α-to-1 multiplexer selects the contents of the state-cell corresponding to that character. The output match vector of width ω carries the information of patterns matched in that state. Each bit in the output-cell indicates whether or not the corresponding pattern is found in the database. In the aforementioned example the values of S, L, α and ω are 4, 4, 26 and 4 respectively. The value of SL is 16 but due to overlapping of certain parts of the patterns the depth of the block RAM is 10 and its width is 108 bits.

3.1.4 Hardware architecture of search engine

As described in section 3.1, the search function and the node operations of the FSM are the most time-consuming tasks in the Aho-Corasick algorithm. For efficient design, these tasks are the best suited for acceleration in hardware. A Search Engine is designed for the AC FSM and the detailed hardware architecture is shown in Fig. 2 (c). Initially, the block RAM in the Search Engine is configured with the FSM table data by enabling memory write and memory store signals. Later, the search operation is started by enabling and disabling the search and memory write signals, respectively. Each character from the database is read and the corresponding part of the memory output, which is a state-cell, is selected by the multiplexer. The next state address of the memory is hold by a State Register at the output of multiplexer, which in turn is generated by the state-cells. The Output Match Vector, which is the pattern match column, indicates the strings that are matched in the respective states. A flag is raised whenever a pattern is successfully matched.

In the next section, we use the block RAM based search engine of ACA FSM explained here and propose the architecture for accelerated and reconfigurable string matching.

3.2 Hardware-Software Codesign based Single Core Architecture for String Matching

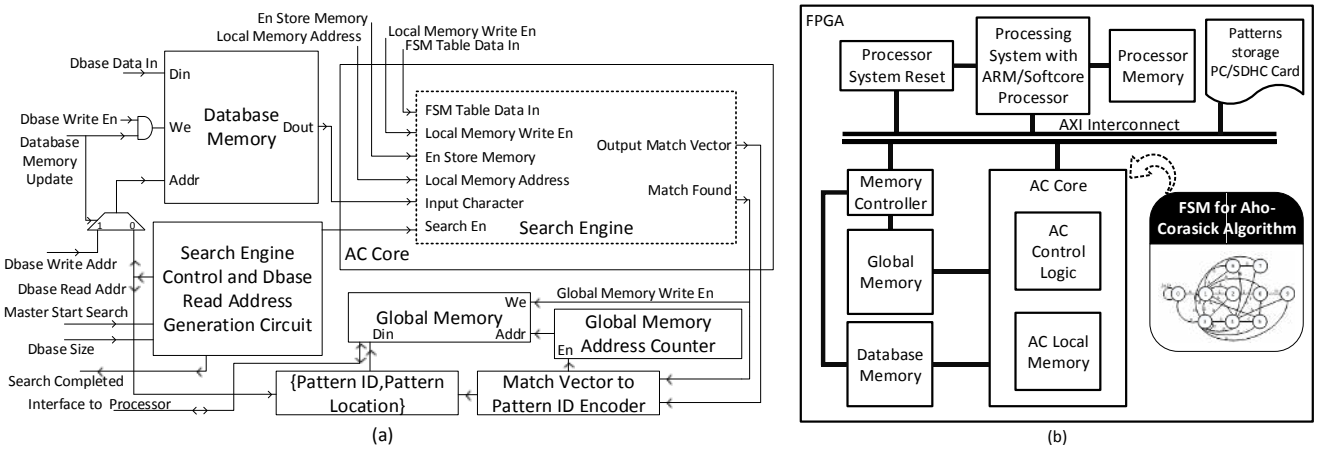

Fig. 3. (a) Hardware architecture of single core (b) System level architecture for hardware-software codesign based Aho-Corasick algorithm using the single core.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCBB.2018.2885296, IEEE/ACM Transactions on Computational Biology and Bioinformatics

The detailed hardware architecture for a single core is presented in Fig. 3 (a). The Search Engine of section 3.1.4 is instantiated inside an AC Core module. The module receives the FSM table data for the patterns and the associated signals to control the local memory operations from the processor or top module where the single core is instantiated. The Search Engine Control Logic generates the necessary signals to control the search operation in the AC core and raises a flag for the processor whenever the search operation is completed on the database. A Database Memory (DM) is used to store the text before the search begins and it feeds the AC Core with the input characters. The Dbase (database) Read Address Generation Circuit (DRAGC) is used to generate the address for the DM during the searching operation. The bits of Output Match Vector of the AC Core indicates the patterns that are found in a given state and they are converted to the corresponding pattern identification numbers in the Encoder. The Encoder generates the address for the Global Memory with the help of an address counter while the concatenation circuit generates the memory contents. The memory input is obtained by clubbing pattern ID and the corresponding location obtained from DRAG circuit. An interface is provided with the Global Memory to read the search results.

System level architecture for the proposed reconfigurable string matching using hardware-software codesign approach is presented in the Fig. 3 (b). For ease of understanding, various signals and control circuits in the above description are clubbed to form blocks representing their corresponding operation. A processing system is formed using an ARM or softcore processor. In the proposed codesign, software part of the system is implemented using the processor. On-chip or on-board DDR memory called as Processor Memory is connected to this processor for storing the compiled software instructions. Due to the limitation of on-chip memory, DDR memory is suitable for programs that produce large memory footprint. A synchronous reset to the entire system is provided by Processor System Reset (PSR) block. Signals from PSR are used to reset all the hardware blocks in the system to initial states. The Memory Controller facilitates a communication link between the processor and memory modules. Advanced Extensible Interface (AXI) memory controller is used for AMBA AXI interconnect systems.

The AC Core block performs string matching operation. Memory based implementation of FSM shown in Fig. 2 (c) is realized using the Local Memory and Control Logic. Depending on the contents of the state register, which holds the value of state in ACA FSM, a particular row is activated in the block RAM which is used to realize Local Memory. The AC Core reads characters from the database and from this row a state-cell value corresponding to the character is selected by the multiplexer. The output of the multiplexer which holds the next state is fed to the state register. Output match vector in the activated row shows the patterns that are matched. Besides taking control actions in the AC Core, the Control Logic keeps track of the counter for the characters and the output match vector. The entire system is residing on an FPGA. Off-chip memories like external hard disc drives, memory cards, flash drives or any other portable memories are used to store large databases.

Software program of the proposed hardware-software codesign architecture, is designed to read the patterns to be searched from an external Secure Digital High Capacity (SDHC) memory card or come from a host machine. These patterns form a set Si. Next, ACA FSM is generated for the corresponding patterns. A memory table is created for the FSM by the software and it is transferred to the Local Memory of the AC Core using the Control Logic. String matching operation is performed on the Database Memory by the AC Core and the results of the search operation are updated to the processor by writing to Global Memory. It can be seen that the AC Core is substituted for AC algorithm and this block acts as a hardware accelerator.

The number of patterns in Si that can be searched in a single pass is limited by the size of Local Memory. For a large number of patterns, an external memory chip is used to replace the Local Memory. In addition, as the proposed system has the feature of runtime reconfiguration, this feature can also be used as an alternate way to tackle this issue. In such cases of a large number of patterns, if the system can search ω patterns in one pass then initially an ω number of patterns are searched followed by the next ω patterns. A direct access to database and global memories from the AC Core, removes the latency in communication and also the overload of memory operations on the software. Also, as soon as the FSM table is stored, AC Core can perform string matching without the intervention of software for memory operations.

3.3 Hardware-Software Codesign based Multiple Core Architecture for String Matching

Due to the ever growing need of computational demand, alternate approaches including parallel computing and multi-core processing find an essential place in computational bioinformatics. Many-core or multi-core platforms are used to speedup underlying algorithms of various bioinformatics applications. Speedup is achieved using multithreading in multi-core CPUs [13] and network-on-chip enabled many-core platforms for sequence analysis and phylogeny reconstruction [17]. In these works, features of operating systems are employed for dividing the task into smaller fragments and speedup is achieved by running the fragments in parallel. This approach requires multi-tasking operating systems and leads to complex programming. In this section, we propose a multi-core architecture using hardware-software codesign for string matching and achieve speedup by eliminating the need for such complex operating systems.

The proposed multi-core architecture for string matching is presented in Fig. 4. Various advantageous features are achieved by the proposed multi-core architecture which include completing the search in lesser time by dividing the database, ability to simultaneously configure AC Cores with patterns of different types, selective powering of individual AC Cores, etc. The multi-core architecture uses an ARM or softcore processor and a Master

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCBB.2018.2885296, IEEE/ACM Transactions on Computational Biology and Bioinformatics

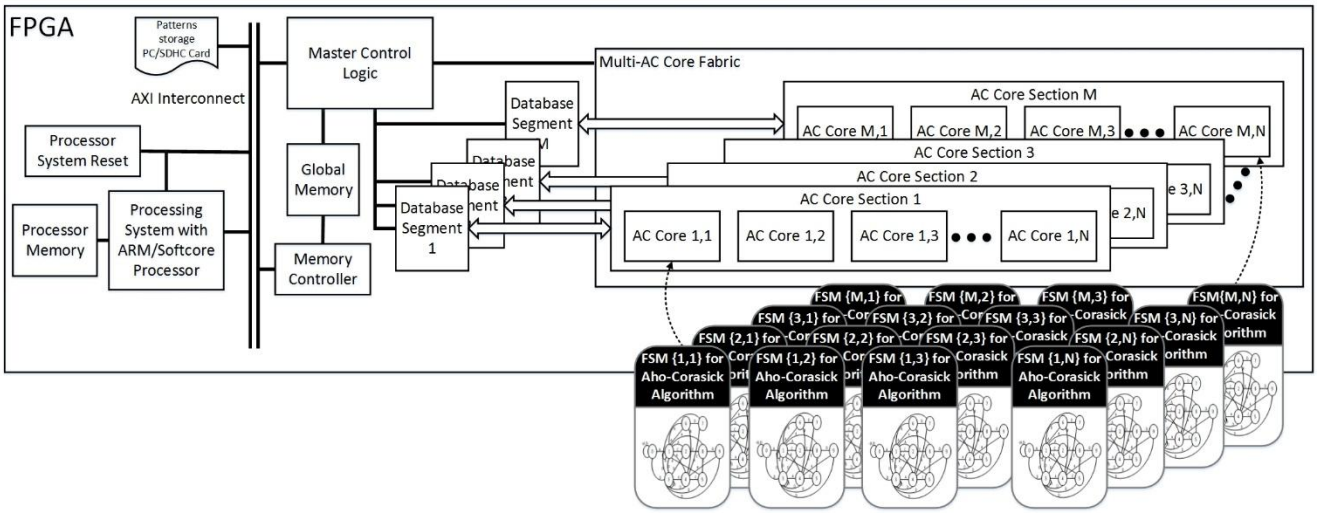

Fig. 4. System level architecture for hardware-software codesign based Aho-Corasick algorithm using the configurable multi-cores. Here the number of AC Cores in every section varies from 1 to N while the number of AC Core Sections varies from 1 to M. The number of sections and cores vary depending on the number of patterns, their size and type of database. This leads to various configurations of the architecture.

Control Logic in a similar fashion as the single core architecture shown in Fig. 3 (b). In this architecture, N number of single AC Cores are connected in parallel and are collectively called as an AC Core Section. For searching large number of patterns, instead of using a single core we use N cores fitted with fewer number of patterns to obtain the advantage of memory reduction as discussed later. In a section, each individual AC Core is selectively powered on and configured with an AC FSM capable to detect a maximum number of ω patterns. Database Segment is a partial or whole database in which search is performed and each section is connected to a segment.

In the proposed architecture, depending on the number of sections, database segments and their arrangement there are four possible types of configurations.

Configuration 1

In the first configuration, a single section is connected to a single database segment. In this configuration, the system can identify patterns of N Aho-Corasick FSMs and a total number of $N\omega$ patterns can be searched in the database. This configuration is useful to search large number of patterns by employing cores fitted with a fewer number of patterns. On a careful analysis, discussed in section 5.1, it is seen that to search $N\omega$ patterns using N cores, each with ω patterns, is memory saving than to use a single core with $N\omega$ patterns. This configuration allows to search multiple sets of patterns in parallel and reduces the searching time compared to the repetitive use of a single core. For example, to search 256 patterns using a 32 patterns core, 8 such cores are required. Memory is saved while designing a core with 32 patterns than a core with 256 patterns.

Configuration 2

In the second configuration, the number of patterns to search is increased by a different arrangement. If the AC Core Sections are duplicated M times and are connected to the same database, then such a configuration leads to identify patterns from MN FSMs and $MN\omega$ patterns can be simultaneously searched in the database. In our system all the AC Core Sections combined together are termed as multi-AC Core Fabric. In this configuration, each individual AC Core in every section is configured with different ω number of patterns and the same database segment is connected for all the M sections. The same functionality can be obtained if the number of cores in the first configuration is extended from N to MN cores. This configuration allows for modularity and reduces the design time by employing the same design of the AC Core Section repeatedly. For example, to search 256 patterns using a section with 8 cores each with 32 patterns, 2 such sections are required.

Configuration 3

In the third configuration, the multi-AC Core Fabric that contains multiple sections is connected to a database divided into smaller parts called segments. Here each section is configured with patterns from same N Aho-Corasick FSMs. This configuration speeds up the search process by searching the database parallelly. If the database of size D is divided into M segments, each of D/M size, then there is M times gain in search time. For example, to search 128 patterns in a database of 100 MB, 2 sections each with the same 128 patterns are used to search in 2 segments of 50 MB.

Configuration 4

The fourth configuration is similar to the third configuration except that each section is configured with patterns from different FSMs. In this configuration, the segments contain databases of different types. The N Aho-Corasick FSMs in each section may be same or different. This configuration allows to search different types of databases for multiple patterns at once. This configuration can be useful for a simultaneous search of different types of patterns in their corresponding databases. For example, to simultaneously search 128 patterns in two different types of databases each of 50 MB, 2 segments and 2 sections are used. A practical scenario would be to search 128 peptide

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCBB.2018.2885296, IEEE/ACM Transactions on Computational Biology and Bioinformatics

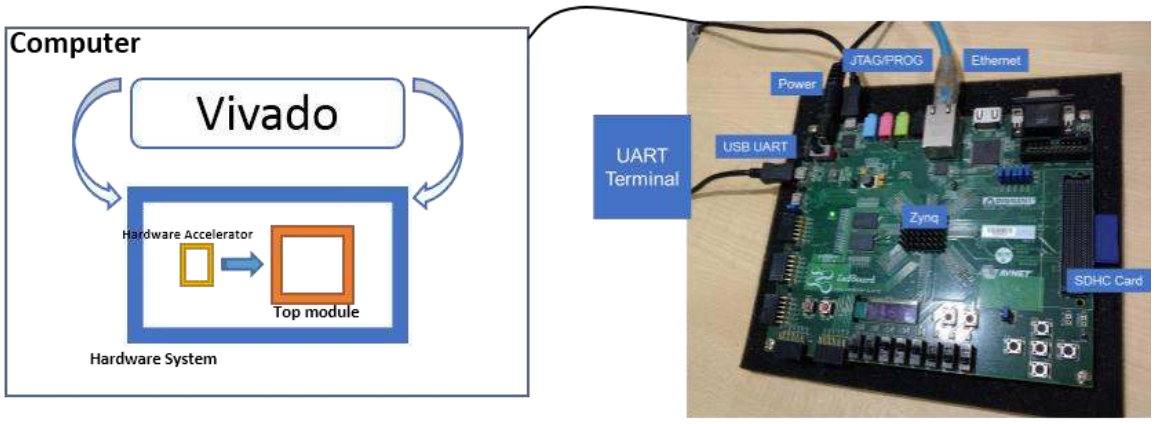

Fig. 5. Prototyped system using Avnet Zedboard

patterns of unknown origin/s in protein databases of different organisms.

3.4 Application of Proposed Methodology for String Matching in Protein Identification

Hardware accelerated solutions for protein identification are used to address the bottlenecks in the computational bioinformatics pipeline created as a result of the excessively rising number of proteins [2], [8], [11], [12], [29]. To demonstrate our proposed methodology, we use protein identification which is a fundamental step in protein sequence analysis. In protein identification using tandem mass spectrometry, one of the most common procedures, peptide fragments obtained by mass spectrometry are matched against large protein databases which is similar to the operation of string matching in a text [2], [8], [11], [12], [29]. Peptides that are obtained from the proteins have grown into millions in number and in addition semi or non-specific digestion has multiplied the number of peptides by many times [29]. To scan databases for these peptides and identify a protein accurately, high speed methods are necessary which can be fulfilled by the proposed methodology. This benefits the discipline of disease biomarker identification and aid disease diagnosis and prognosis [22].

Peptide fragments are the patterns to be searched and these are used to create AC FSM as explained in section 3.1.2. Proposed design is useful to search proteome databases at high speed and simultaneously supports runtime reconfiguration with peptides patterns. FASTA format is used to store the proteomes in database memory. Simple binary coding is used to represent different amino acids and additional symbols of the FASTA format. For protein identification AC Core is designed to keep track of protein ID, peptides matched and locations of peptides in the protein. These results are stored in global memory and updated to software where software does post-processing like the ranking of proteins according to their scores.

4 System Implementation on FPGA

Zynq SoC FPGA device with an ARM processor or any FPGA with a softcore processor can be useful to accomplish the proposed reconfigurable hardware-software codesign methodology. The Zynq SoC FPGA has a reconfigurable hardware fabric and a programmable processor on the same chip making it an ideal choice. To verify our proposed architecture, we have used Xilinx Vivado Design Suite software tools and Avnet Zedboard development board. In the system implementation, patterns are read from an external SDHC memory card. An AC FSM is generated for these patterns in software and converted to a table as explained earlier in section 3.1. This FSM table is stored in the local BRAM memory with the help of Control Logic. The AXI BRAM controller is the hardware that is used as the memory controller for the Zynq device in the design. For verification, we use the on-chip memory for storing the database.

The proposed system is implemented by following the embedded system design flow for Vivado tool [31]. The AC Core is synthesized independently using the Vivado synthesis tool. For default constraints, the maximum frequency at which the core can run is 316.776 MHz. Higher frequency rates are possible by constraining the synthesis tool more stringently at the cost of increased area and resources. Avnet Zedboard that has an on board XC7Z020 FPGA device is used for the verification purpose. ARM Cortex-A9 MPCore CPU available on this FPGA is used as the processing system. The AC Core is packaged as an IP and is interfaced to the ZYNQ7 processing system via an AXI interconnect.

A prototype diagram of the system is shown in Fig. 5.

TABLE 2
Resource Utilization on FPGA by Single and Multi Core Systems

Resources	Single Core	Multi Core	Available
LUT	3318	7380	53200
LUT memory	182	182	17400
Flip-flop	4662	11725	106400
Block RAM	32.5	112	140
BUFG	1	1	32

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCBB.2018.2885296, IEEE/ACM Transactions on Computational Biology and Bioinformatics

TABLE 3
Power Consumption of Single and Multi Core Systems

Resource Unit	Single Core	Multi Core
Clocks	0.013	0.023
Slice Logic	0.005	0.007
Signals	0.007	0.01
Block RAM	0.02	0.039
PS7	1.527	1.527
Static Power	0.162	0.164
Accelerator	0.023	0.058
Total	1.734	1.77

Note: All units are in watt

The hardware accelerator, either single AC core or multi-AC core, is synthesized and then packaged into an IP using Vivado tool. The custom IP is instantiated in a top module along with the processing system and other required blocks as depicted in Fig. 3 and Fig. 4. The complete hardware system is synthesized and implemented. A bitstream file is generated and it is used to configure the Avnet Zedboard using JTAG. A C program is written in SDK tool of Vivado suite following the algorithmic flow described in Table 1. Using hardware drivers, the C program calls the hardware accelerator for matching patterns, stored on the SDHC memory card, in the databases which are also stored on the memory card. Zedboard configuration with the bitstream file is a one-time process as the hardware logic is not required to modify. String matching results can be communicated through the UART port or stored on the SDHC card.

Post-implementation results showing resources utilized by both single and multi-AC core systems are presented in Table 2. From the resources available on FPGA 3318 LUT (6.24%), 4662 FF (4.38%) and 32.50 BRAM (23.21%) are consumed by the single AC core system and total on-chip power consumption of the whole system is 1.734 watt. For multi-AC core system of $M=2$ and $N=4$ values, 7380 (13.87%), 11725 FF (11.02%) and 112 BRAM (80%) are consumed and 1.77 watt is the power consumption of the system. Vivado power analysis is used to estimate the power consumption and is presented in Table 3. Approximately 86% to 88% power is consumed by the processing system (ARM processor). The hardware accelerator in single core and multi core systems consumes 0.023 watt and 0.058 watt respectively which is very small in comparison with CPU or GPU based string matching systems.

Each synthesized and implemented system is exported to the Xilinx SDK software tool along with their corresponding bitstream file. A C program is written by following the algorithm described in Table 1. As the AC Cores are interfaced to the processing system as custom IPs, driver functions that are needed to communicate with these custom IPs are also written in a separate C header file. For interested readers, more details about embedded system design flow and custom IP packaging and interface can be found in [31]. Due to the limited BRAM resources on the FPGA we search the entire databases in batches. External memories can also be used in place of FPGA BRAM and the entire database can be searched in a single pass. When off-chip memory is used for storing database, driver functions to communicate with the external memory should be written.

TABLE 4
Dependency of Configuration Time on Number of Patterns in Core

Number of Patterns in Core	Maximum Memory Required (kB)	Reconfiguration Time (ms)	Number of Times Core Reused	Search Complete Time (ms)
4	2.725	39.73	512	6154
8	6.328	66.72	256	3077
16	14.648	123.99	128	1538
32	34.219	247.70	64	769
64	82.031	602.13	32	385
128	206.250	1337.67	16	192

Note: Number of total patterns is 2048 and size of the database to search is 1.35 MB.

5 Evaluation

In this section, we evaluate the performance of the AC core hardware and the architectures designed using the proposed methodology. Firstly, for a different number of patterns, we analyze the memory requirement of the AC core. Then we report reconfiguration time of the AC core obtained from the hardware setup. Later we evaluate the time required for searching databases using the proposed architectures and compare the performance with the existing literature.

5.1 Analysis for Number of Patterns per AC Core

To study memory consumption by AC cores, a mathematical analysis is performed for the memory requirement versus the number of patterns per core. For large number of patterns, the string matching operation is performed in batches of ω patterns, where ω is the number of pat-

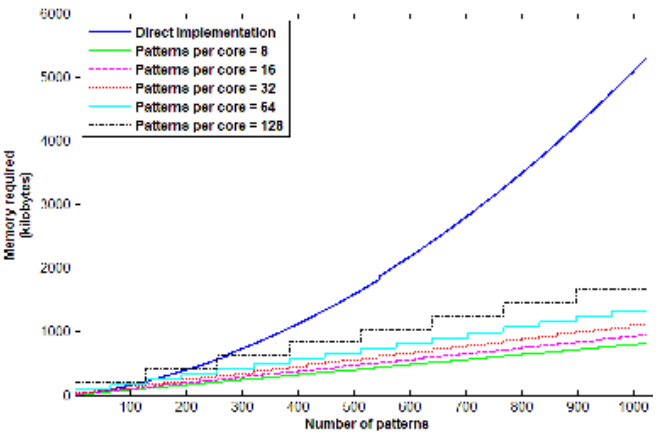

Fig. 6. Analysis for the number of patterns in a core. Memory requirement for matching Ω patterns varies depending on the number of patterns in the core.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCBB.2018.2885296, IEEE/ACM Transactions on Computational Biology and Bioinformatics

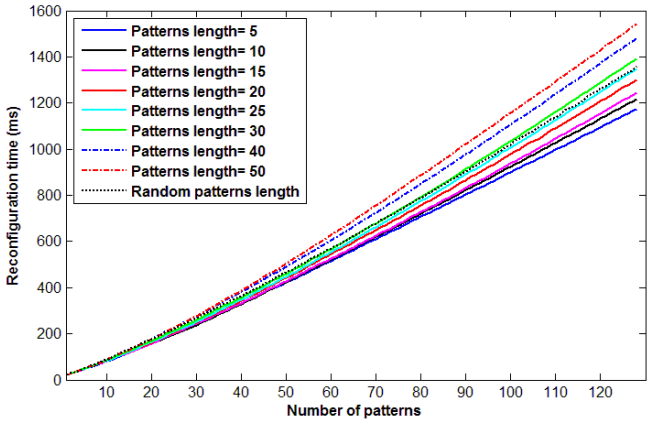

Fig. 7. Time required to reconfigure the system with patterns. Here dependency of reconfiguration time on number of patterns is plotted for varying length of patterns.

terns per core. For matching Ω total number of patterns where $\Omega > \omega$, the core has to be replicated or reused Ω/ω times. As explained earlier in section 3.1.2, width of each register is $\alpha \log_2(SL) + \omega$ bits. Fig. 6 shows the memory required for implementing patterns of various numbers using a core fitted with a smaller number of patterns. In the figure, direct implementation refers to realizing a string matching system for Ω patterns using a single core and in this case, all the Ω patterns are stored in a single FSM. For an example, to realize 128 patterns using 8 patterns per core, the core has to be replicated or reused 16 times and in direct implementation a single core can search all 128 patterns.

From the figure, it is evident that lower amounts of memory are required when a core with a smaller number of patterns is chosen. This advantage is employed in the multi-AC core system. The effect of reusing cores for realizing Ω patterns in terms of reconfiguration time is studied experimentally for the proposed single-core design. In Table 4 these results are presented for cores fitted with 4, 8, 16, 32, 64 and 128 patterns per core. Here we have chosen Ω as 2048 patterns and a database of 2650 proteins of 1.35 MB total size is used for searching. Each core is reused as many number of times as required to cover Ω (2048) patterns. The database has to be rescanned every time whenever the core is configured with new patterns. It is observed that search time increases linearly with the number of times the core is reused. Since the data bus size and word alignment in the processor is of 32 bit, we choose 32 patterns per core as an optimum number. In addition to this, 32 bit word size is also useful to reduce the overload of transferring data since lower sizes require more number of transfers for the same amount of data.

5.2 Performance on Reconfiguration Time

When patterns are changed, an AC FSM is created and the cores are reconfigured with new FSM tables. To examine the proposed system's use for real-time reconfiguration, we have studied the effect of the number of patterns and their size on reconfiguration time of an AC core. Here patterns of varying total sizes are considered and the reconfiguration time of AC core obtained from hardware is plotted against the number of patterns in Fig. 7. These results are obtained for a fabric clock of 100 MHz. From the figure, it is evident that there is a linear relation between reconfiguration time and the number of patterns; the reconfiguration time of a core increases linearly with the number of patterns in it. It is seen that on an average, the reconfiguration time is about a few hundred of milliseconds value and thus our proposed methodology is worth for employing in applications that demand real-time or on-field reconfiguration. To the best of our knowledge, this work is the first of its kind where the time required for reconfiguration of a hardware assisted string matching system for bioinformatics applications is reported in the literature.

5.3 Search Time for Single Core System

In this section, firstly we evaluate single core based string matching system regarding search time, throughput, and improvement over software approaches for string matching. Next, we perform string matching in large proteomic databases to validate the practical use of the proposed system. Later, we compare our proposed methodology with similar approaches present in the literature.

TABLE 5
Features of Patterns and Search Time for Single Core Architecture

Type of Configuration	Type of Dbase	#patterns[1]	#bytes[2]	Max (l)[3]	Average (l)[4]	σ[5]	Software (PC) ms	Software (ARM) ms	Proposed ms
Single Core	Dbase1	31.6	397.4	43.4	12.834	9.244	1511.3	9339	418.120
	Dbase2	58.2	708.5	44.5	12.339	8.295	1912.7	10552	751.447
	Dbase3	33.2	496.7	50.6	16.144	11.044	1590.1	9826	504.472
	Dbase4	35.4	525.9	53.6	16.586	13.064	1872.7	10977	526.662
	Dbase5	32.7	430.9	35.6	12.692	8.145	3341.2	18341	673.161
	Dbase5	32.7	430.9	35.6	12.692	8.145	3341.2	18341	673.161

Dbase = database, Dbase1 = 13786.526, Dbase2 = 14545.275, Dbase3 = 14954.893, Dbase4 =16944.043, Dbase 5 = 32272.287 (all values in kB)

[1] number of patterns, [2] total number of bytes in all patterns, [3] maximum length pattern, [4] average length of pattern, [5] standard deviation.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCBB.2018.2885296, IEEE/ACM Transactions on Computational Biology and Bioinformatics

TABLE 6
Search Time for String Matching in Large Proteomic Databases

Peptide set	AC Core Reconfiguration Time (ms)	Expected Search Time[1] (ms)	Actual Search Time[2] (s)
Set #1	161.720	171.188	257.721
Set #2	207.004	216.475	257.643
Set #3	203.400	212.867	257.757
Set #4	98.222	107.689	257.758
Set #5	1890.387	1966.123	2061.843
Set #6	2023.072	2098.809	2062.241
Set #7	2310.427	2386.164	2061.847
Set #8	3296.980	3372.716	2062.626
Set #9	1880.969	1956.717	2062.317

Note: [1] Search time without considering the data transfer overload from SDHC card to FPGA, [2] Search time including the time required to transfer data from SDHC card to FPGA, Set #1 to Set #4 has 32 patterns and are obtained by digestion of proteins, Set #5 to Set #8 has 16, 32, 64, 128 length patterns respectively and in each set there are 256 patterns, Set #9 has 256 patterns of random length

5.3.1 Performance Evaluation of Single Core System

To test the proposed singe AC core system for protein identification, we have used UniProt proteomics data [30]. To demonstrate on-field reconfiguration, we have considered data of five different organisms. As explained previously in 5.1, a single core system which can search 32 patterns is designed. In the experiments, different sets of proteins in each database is chosen randomly and the databases are searched for the peptides obtained from the digestion of these proteins. PeptideMass [10], an online tool for enzymatic cleavage of proteins, is used to digest these selected proteins. The peptides obtained after proteins digestion are stored in different patterns files with their corresponding identifier as file names. These peptides are patterns for the string matching and are searched in the proteome databases. We have also tested the improvement of proposed hardware-software codesign methodology over a software only Aho-Corasick string matching algorithm running on a workstation. The workstation has Windows 7 operating system running on Intel Xeon E5-2650 v2 CPU @ 2.60 GHz with 8 GB RAM. A similar software version of the ACA is also implemented using the ARM processor available on the Zynq XC7Z020 FPGA. This implementation demonstrates a microprocessor or microcontroller tailored embedded system solution for reconfigurable string matching. A clock of 100 MHz is used for running the hardware and this facilitates with a constant throughput of 800 Mbps. The results obtained are shown in Table 5. Only a few sets of results are given in the table due to space limitation. These results are obtained after calculating average values for multiple test cases in every database. On an average, there is a 4 times improvement in search time by the proposed hardware-software codesign methodology over software only version running on the workstation and 23 times improvement over the software running on the ARM processor. While calculating the gain in speed, we have considered the time required for multiple reconfigurations and also the time required for multiple searching of database. If only the time required for searching a single batch is considered, then the improvement over speed is 13 and 70 times for software running on the workstation and ARM respectively.

TABLE 7
Comparison of Proposed Methodology with Similar Method

Comparison metric	Lei [32]	Proposed
Speedup vs CPU	2X	4X
Speedup vs ARM	5X	23X
Hardware Accelerator Power	60 mW	23 mW
Total Power	1.368 W	1.734 W
Type of pattern searching	Single	Multiple
Host PC requirement	Yes	No

Since the synthesis results permit to use clock up to 316.776 MHz for the default constraints, by using clocks of higher speed search time can be improved further. In comparison with the average throughput value of 522.88 Mbps obtained by Dandass et al [8] for 100 MHz clock, our architectures have a throughput of 800 Mbps for the same clock. In addition to real-time reconfiguration, these results imply the proposed methodology is 1.5 times faster.

5.3.2 String Matching in Large Proteomic Databases

To examine the practical applicability of the proposed methodology, we constructed a protein database comprised of 10 different primate animals [30]. Each database has a varying number of proteins and the total number of proteins in the concatenated database is 297,293. The database size is ~153 MB and it has 159,654,352 amino acids. Randomly selected proteins are digested using PeptideMass [10]. The maximum number of peptides obtained from each protein is limited to 32. Table 6 summarizes the results. Due to space limitation, only a few test cases are presented in the table.

Next, to study the effect of the number of patterns and their length, we created a set of simulated patterns. The number of patterns in each test case is 256 while their

TABLE 8
Comparison of Proposed Methodology with Software Methods

Pattern Length	Faro [33]		Proposed		Speed Gain
	Time	Time/Pattern	Time	Time/Pattern	
4	2320	5.8	42.253	0.16505	35.14
8	2590	6.475	42.427	0.16573	39.07
16	1910	4.775	42.269	0.16511	28.92
32	1560	3.9	42.385	0.16557	23.55
64	1520	3.8	42.676	0.16670	22.80
128	1550	3.875	43.722	0.17079	22.69
256	1210	3.025	43.921	0.17157	17.63

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCBB.2018.2885296, IEEE/ACM Transactions on Computational Biology and Bioinformatics

TABLE 9
Features of Patterns and Search Time for Multi-core Architectures

Type of Configuration	Type of Dbase	#patterns[1]	#bytes[2]	Max (l)[3]	Average (l)[4]	σ[5]	Software (PC) ms	Software (ARM) ms	Proposed ms
Multi-core Configuration 1	Dbase1	87	1080	36	12.384	7.879	1504.0	10390	814.067
	Dbase2	115	1433	44	12.605	7.942	1582.0	11009	995.799
	Dbase3	57	1168	70	20.738	15.020	1663.0	10645	594.040
	Dbase4	74	1407	90	19.239	16.833	2318.0	11915	767.319
	Dbase5	87	1125	62	12.754	7.844	4277.0	20472	967.080
Multi-core Configuration 2	Dbase1	202	2621	57	13.026	8.902	1953.0	11479	1936.286
	Dbase2	256	3537	61	12.701	8.430	2082.0	12274	2398.845
	Dbase3	206	3215	75	17.209	11.988	2130.0	20460	1984.032
	Dbase4	212	3424	90	17.658	13.754	2428.0	22659	2085.001
	Dbase5	179	2212	62	12.394	8.079	4281.0	24637	1988.683
Multi-core Configuration 3	Dbase1	87	1080	36	12.384	7.879	1504.0	10390	753.681
	Dbase2	115	1433	44	12.605	7.942	1582.0	11009	930.794
	Dbase3	57	1168	70	20.738	15.020	1663.0	10645	527.840
	Dbase4	74	1407	90	19.239	16.833	2318.0	11915	692.512
	Dbase5	87	1125	62	12.754	7.8449	4277.0	20472	829.604
Multi-core Configuration 4	Dbase1 & Dbase2	202	2513	44	12.494	7.911	1953.0	11479	1689.193
	Dbase3 & Dbase4	131	2575	90	19.989	15.927	2082.0	12274	1228.972

**Dbase = database, Dbase1 = 13786.526, Dbase2 = 14545.275, Dbase3 = 14954.893, Dbase4 =16944.043, Dbase 5 = 32272.287 (all values in kB)*
[1] number of patterns, [2] total number of bytes in all patterns, [3] maximum length pattern, [4] average length of pattern, [5] standard deviation.

length is varied from 16, 32, 64 and 128. Finally, a test case comprising of 256 patterns of random lengths is also considered. For each case, multiple datasets are used for running the proposed string matching methodology. The results obtained are also presented in Table 6. In each case, only the average values of the results obtained for multiple datasets is presented in the table. From the table, it is evident that the proposed system is applicable for searching large sized databases within a reasonable time.

5.3.3 Cross Examination with Similar Approaches

To compare the performance of the proposed methodology, we consider evaluating with similar work reported in the literature [32], [33].

Lei et al proposed a KMP algorithm based accelerator for string matching and implemented using Avnet Zedboard [32]. As the actual time for searching is not available, we use normalized speedup for comparing speed improvement. The comparison is presented in Table 7. The proposed method is 4X faster in comparison with a software running on CPU while Lei et al is 2X faster and in comparison with ARM version, proposed method is 23X faster while Lei et al is 5X faster. Overall, the proposed methodology is 2X-4X faster than Lei et al. Except for total power, the proposed methodology outperforms Lei et al in all metrics since the former has higher power consumption as the processing system (PS7) consumes more power than the PS in Lei et al.

Faro et al presented an extensive survey of exact string matching algorithms [33]. Experimental results obtained by running various algorithms implemented in C program are also presented. For experimentation, a 1.66 GHz PC with Intel Core2 processor and 2GB RAM is used. A protein sequence of 3,295,751 length is taken from Protein Corpus for the experimental purpose (http://data-compression.info/Corpora/ProteinCorpus/). We have used the same database and performed string matching using the proposed methodology. Patterns of varying lengths 4, 8, 16, 32, 64, 128 and 256 are considered. The results are tabulated in Table 8. Faro et al considered 400 patterns. Due to limited BRAM resources, we choose 256 patterns. The search time for varying lengths of patterns and search time per pattern are examined. For every algorithm implemented in C program, the best time reported in Faro et al is given in Table 8. On average the proposed methodology is 27 times faster than the algorithms surveyed in [33]. Here all the time units are in second. Time/Pattern indicates the average time taken to search one pattern in the database.

5.4 Search Time for Multi Core System

A similar flow that is followed in the single core system is also followed for testing the proposed multi-core architecture shown in Fig. 4. As mentioned earlier, there are four configurations for the proposed architecture. In the first configuration, we have used four cores in the section and this configuration can search a maximum of 128 patterns in a single pass. We can search more than 128 patterns by

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TCBB.2018.2885296, IEEE/ACM Transactions on Computational Biology and Bioinformatics

reconfiguring the cores with the next batch of patterns. In the second configuration, we have used two sections and each section contains four AC cores. Overall, there are eight cores and each core is able to search all the patterns from its corresponding FSM with which it is configured. This configuration can search a maximum of 256 patterns in the database segment in a single pass. In the third configuration, we have used two sections, each containing same four AC cores. These sections are connected to multiple database segments obtained by dividing the whole database. This configuration can search a maximum of 128 patterns and can approximately half the search time that is obtained by the first configuration. In the fourth configuration, we have used two sections with four cores in each section and these are connected to two different databases. This configuration can search two sets of 128 patterns in their corresponding databases. The experimental results obtained for all the four configurations are tabulated in Table 9. Due to space limitation, only a few sets of average results are given in this table. From the results, it can be inferred that for all the different types of configurations that emerge from various requirements, multiple core system is useful and in comparison with software running on workstation and ARM, they outperform with a speed improvement of 2X-20X.

6 Conclusion

In this paper, we proposed an accelerated and real-time reconfigurable methodology for string matching using hardware-software codesign. Using state of the art FPGAs, we have proposed a complete system-on-chip solution for applications that require accelerated as well as real-time reconfigurable string matching and it is verified at the string matching stage of protein identification. By using the proposed methodology, there is a 4X and 1.5X-4X improvement of search speed in comparison with state-of-the-art software and hardware accelerators available in the literature and at the same time, the methodology has real-time reconfiguration feature. The proposed methodology achieves reconfigurable string matching and also bypasses the use of proprietary tools required for reconfiguring the system by patterns changing with time. Experimental results show that we are able to achieve real-time reconfigurable string matching with an average value of the order of milliseconds. The implemented systems have a constant throughput rate that is decided by the clock used for the hardware and it can search databases at a throughput of approximately 800 Mbps for a clock of 100 MHz value.

Acknowledgment

In this work, V. Y. Gudur was supported by the Visvesvaraya PhD Scheme for Electronics & IT by the Ministry of Electronics & Information Technology (MeitY), Government of India. A. Acharyya was supported by Visvevaraya Young Faculty Fellowship funded by MeitY, Government of India. All the software tools are supported under Special Manpower Development Programme for Chips to Systems funded by MeitY, Government of India. The authors' would like to thank the anonymous reviewers for their many insightful comments and suggestions that improved the quality of this paper. A. Acharyya is the corresponding author.

References

[1] A. Aho and M. Corasick, "Efficient String Matching: An Aid to Bibliographic Search", *Communications of the ACM*, vol. 18, no. 6, pp. 333-340, 1975.

[2] A. Alex, M. Dumontier, J. Rose and C. Hogue, "Hardware-Accelerated Protein Identification for Mass Spectrometry", *Rapid Communications in Mass Spectrometry*, vol. 19, no. 6, pp. 833-837, 2005.

[3] S. Aluru and N. Jammula, "A Review of Hardware Acceleration for Computational Genomics", *IEEE Design & Test*, vol. 31, no. 1, pp. 19-30, 2014.

[4] J. Arram, T. Kaplan, W. Luk and P. Jiang, "Leveraging FPGAs for Accelerating Short Read Alignment", *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, vol. 14, no. 3, pp. 668-677, 2017.

[5] D. Benson, M. Cavanaugh, K. Clark, I. Karsch-Mizrachi, D. Lipman, J. Ostell and E. Sayers, "GenBank", *Nucleic Acids Research*, vol. 45, no. 1, pp. D37-D42, 2016.

[6] M. Brudno, M. Chapman, B. Göttgens, S. Batzoglou and B. Morgenstern, "Fast and Sensitive Multiple Alignment of Large Genomic Sequences", *BMC Bioinformatics*, vol. 4, no. 1, p. 66, 2003.

[7] P. Chen, C. Wang, X. Li and X. Zhou, "Accelerating the Next Generation Long Read Mapping with the FPGA-Based System", *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, vol. 11, no. 5, pp. 840-852, 2014.

[8] Y. Dandass, S. Burgess, M. Lawrence and S. Bridges, "Accelerating String Set Matching in FPGA Hardware for Bioinformatics Research", *BMC Bioinformatics*, vol. 9, no. 1, p. 197, 2008.

[9] T. Dorta, J. Jiménez, J. Martín, U. Bidarte and A. Astarloa, "Reconfigurable Multiprocessor Systems: A Review", *International Journal of Reconfigurable Computing*, vol. 2010, pp. 1-10, 2010.

[10] E. Gasteiger, C. Hoogland, A. Gattiker, S. Duvaud, M. Wilkins, R. Appel and A. Bairoch, "Protein Identification and Analysis Tools on the ExPASy Server", *The Proteomics Protocols Handbook*, pp. 571-607, 2005.

[11] V.Y. Gudur, S. Thallada, A. Deevi, V. Gande, A. Acharyya, V. Bhandari, P. Sharma, S. Khursheed and G. Naik, "Reconfigurable Hardware-Software Codesign Methodology for Protein Identification", *2016 38th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*, Orlando, FL, pp. 2456-2459, 2016.

[12] W. Henzel, C. Watanabe and J. Stults, "Protein Identification: The Origins of Peptide Mass Fingerprinting", *Journal of the American Society for Mass Spectrometry*, vol. 14, no. 9, pp. 931-942, 2003.

[13] D. Herath, C. Lakmali and R. Ragel, "Accelerating String Matching for Bio-computing Applications on Multi-Core CPUs," *2012 IEEE 7th International Conference on Industrial and Information Systems (ICIIS)*, Chennai, pp. 1-6, 2012.

[14] Y. Hu and P. Georgiou, "A Real-Time de novo DNA Sequencing Assembly Platform Based on an FPGA Implementation", *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, vol. 13, no. 2, pp. 291-300, 2016.

[15] H. Hyyrö, M. Juhola and M. Vihinen, "On Exact String Match-

ing of Unique Oligonucleotides", *Computers in Biology and Medicine*, vol. 35, no. 2, pp. 173-181, 2005.

[16] H. Kim and K. Choi, "A Pipelined Non-Deterministic Finite Automaton-Based String Matching Scheme Using Merged State Transitions in an FPGA", *PLOS ONE*, vol. 11, no. 10, p. e0163535, 2016.

[17] T. Majumder, P.P. Pande and A. Kalyanaraman, "On-Chip Network-Enabled Many-Core Architectures for Computational Biology Applications," *2015 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, Grenoble, pp. 259-264, 2015.

[18] C. Maxfield, *The Design Warrior's Guide to FPGAs: Devices Tools and Flows*: Newnes, pp. 153-178, 2004.

[19] M. Michael, C. Dieterich and M. Vingron, "SITEBLAST-Rapid and Sensitive Local Alignment of Genomic Sequences Employing Motif Anchors", *Bioinformatics*, vol. 21, no. 9, pp. 2093-2094, 2004.

[20] S. Palnitkar, *Verilog HDL: A Guide to Digital Design and Synthesis*, Noida, India: Dorling Kindersley (India), pp. 45-52, 2006.

[21] C.S. Rao, K.B. Raju, S.V. Raju, "String Matching Problems with Parallel Approaches-An Evaluation for the Most Recent Studies," *Global Journal of Computer Science and Technology*, vol. 13, 11-C, pp. 9-18, 2013.

[22] Z.J. Sahab, S.M. Semaan, A.S. Qing-Xiang, "Methodology and Applications of Disease Biomarker Identification in Human Serum", *Biomarker Insights*, vol. 2, pp. 21-43, 2007.

[23] M. Santarini, "Zynq-7000 EPP Sets Stage for New Era of Innovations", *Xcell journal*, no. 75, pp. 8-13, 2011.

[24] E. Schadt, M. Linderman, J. Sorenson, L. Lee and G. Nolan, "Computational Solutions to Large-Scale Data Management and Analysis", *Nature Reviews Genetics*, vol. 11, no. 9, pp. 647-657, 2010.

[25] P.R. Schaumont, "The Nature of Hardware and Software," *A Practical Introduction to Hardware/Software Codesign*, 2nd ed., New York, USA: Springer Science+Business Media, pp. 3-30, 2013.

[26] R. Senhadji-Navarro, I. García-Vargas and J.L. Guisado, "Performance Evaluation of RAM-based Implementation of Finite State Machines in FPGAs," *2012 19th IEEE International Conference on Electronics, Circuits, and Systems (ICECS 2012)*, Seville, pp. 225-228, 2012.

[27] J. Teich, "Hardware/Software Codesign: The Past, the Present, and Predicting the Future", *Proceedings of the IEEE*, vol. 100, no., pp. 1411-1430, 2012.

[28] J.G. Tong, I.D.L. Anderson and M.A.S. Khalid, "Soft-Core Processors for Embedded Systems," *2006 International Conference on Microelectronics*, Dhahran, pp. 170-173, 2006.

[29] C. Zhou, H. Chi, L. Wang, Y. Li, Y. Wu, Y. Fu, R. Sun and S. He, "Speeding Up Tandem Mass Spectrometry-based Database Searching by Longest Common Prefix", *BMC Bioinformatics*, vol. 11, no. 1, p. 577, 2010.

[30] The UniProt Consortium, "UniProt: A Hub for Protein Information," *Nucleic Acids Research*, vol. 43, pp. D204-D212, 2015.

[31] "Zynq-7000 All Programmable SoC: Embedded Design Tutorial," *A Hands-On Guide to Effective Embedded System Design*, UG1165 (v2017.3), Xilinx, Inc.

[32] S. Lei, C. Wang, H. Fang, X. Li and X. Zhou, "SCADIS: A Scalable Accelerator for Data-Intensive String Set Matching on FPGAs," 2016 IEEE Trustcom/BigDataSE/ISPA, Tianjin, pp. 1190-1197, 2016.

[33] S. Faro and T. Lecroq, "The Exact Online String Matching Problem: A Review of the Most Recent Results," ACM Computing Surveys, vol. 45 (2), pp 13-42, 2013.

[34] I. A. Bogdan, J. Rivers, J. R. Beynon and D. Coca, "High-performance hardware implementation of a parallel database search engine for realtime peptide mass fingerprinting," *Bioinformatics*, vol. 24, no. 13, pp. 1498-1502, 2008.

[35] A. Zerck, E. Nordhoff, A. Resemann, E. Mirgorodskaya, D. Suckau, K. Reinert, H. Lehrach, and J. Gobom, "An Iterative Strategy for Precursor Ion Selection for LC-MS/MS Based Shotgun Proteomics," *Journal of Proteome Research*, vol. 8, no. 7, pp. 3239-3251, 2009.

[36] R. J. Peace, H. Mahmoud and J. R. Green, "Exact string matching for MS/MS protein identification using the Cell Broadband Engine," *Journal of Medical and Biological Engineering*, vol. 31, no. 2, pp. 99-104, 2011.

Venkateshwarlu Y. Gudur received the BE degree in Electronics and Telecommunication from Walchand Institute of Technology, Solapur and the M.Tech degree in VLSI Design from Shri Ramdeobaba College of Engineering and Management, Nagpur, in 2012 and 2014 respectively. Currently, he is working towards the PhD degree in Microelectronics and VLSI at the Department of Electrical Engineering, Indian Institute of Technology (IIT) Hyderabad, India. His research interests include hardware acceleration in healthcare applications, VLSI architectures, multiprocessor SoC and reconfigurable computing.

Amit Acharyya received the Ph.D. degree from the School of Electronics and Computer Science, University of Southampton, U.K., in 2011. He is currently an Associate Professor with IIT Hyderabad, Hyderabad, India. His research interests include signal processing algorithms, VLSI architectures, low power design techniques, computer arithmetic, numerical analysis, linear algebra, bio-informatics, and electronic aspects of pervasive computing.