ISSN: 2277-3754
International Journal of Engineering and Innovative Technology (IJEIT)
Volume 1, Issue 2, February 2012

A Survey of Parallel Algorithms for Text Matching In Large Databases and Hardware Implementations

K S M V Kumar, S Viswanadha Raju, A Govardhan
ksmvkumar@yahoo.co.in,viswanadha_raju2004@yahoo.co.in,govardhan_cse@yahoo.co.in

Abstract- String Matching plays a vital role in wide range of applications, such as Network Intrusion Detection in packet routing, IP address lookup in routers, cloud computing, DNA analysis, Genome analysis and searching for given text in voluminous of data , similarly to Data Mining is also Motivation factors for the study of Text Retrieval. Due to it's importance, several Parallel Algorithms and Data Structures have been evaluated for Text Matching in Past two decades, but they are insufficient to meet the current technology development in networking, multimedia databases, audio databases and cloud computing .Text matching is deals with the problem of finding all the occurrences of Pattern in a Large Text Data bases . Owing to extensive growth of data base size and network routing complexity, the text Matching software algorithms and Hardware approaches should be modified .This paper focuses on these applications areas and presents a survey of several exact String matching, approximate String matching and k-mismatches problems along with their Hardware Implementations like PRAM computational model ,CAM implementation on FPGA, and Cellular array processor(CAP) to speedup further.

***Keywords*- DNA, Network Intrusion Detection, PRAM, CAM, CAP, Text retrieval and data mining.**

I. INTRODUCTION

TEXT MATCHING APPROACHES IN HARDWARE: The string (Text) Matching problem received much attention in fast two decades. Most extensively Investigated and widely used problem in various applications such as string data base information retrieval systems, Text processing,DNA analysis, web search engine ,Artificial Intelligence, NID ,Keyword blocking, Anti virus,Anti spam, comparing of Two input strings and several other area's such as data comparison, speak recognition, Image processing etc. Therefore, it is necessary to design a parallel Text Matching algorithm to reduce the work load and Implementations of those algorithms using a high speed Hardwares.

The string matching problem can be divided into two major categories, known as exact string Matching and approximate string matching, further subcategory into parallel String Matching and String Matching with K-mismatches. All these problems can be solved by either software-based solutions or Hardware-based solutions .since software-based solutions are slower and less efficient, hardware-based solutions are highly preferred. This section continues with a brief discussion and classification of all early string matching algorithms was covered .To have better understanding of how string matching is done in hardware, the approaches proposed for these applications will be discussed in Section II to Section IX .

Exact String Matching (Single String Matching): One of the earliest exact string matching algorithms is Aho-Corasick algorithm [1]. The algorithm locates all occurrences of any keywords in a text string. It works by constructing a finite state pattern matching machine from the keywords, and then using the pattern matching machine to process the text string in a single pass. The state machine starts with an empty root node. Each pattern to be matched adds states to the machine, starting at the root and going to the end of the pattern. The state machine is then traversed and failure pointers are added to indicate any disconnection between two states. Figure 1 illustrates the state machine for the set of keywords {he, she, his, hers} . The time complexity of Aho-Corasick algorithm is linear in the size of the input.

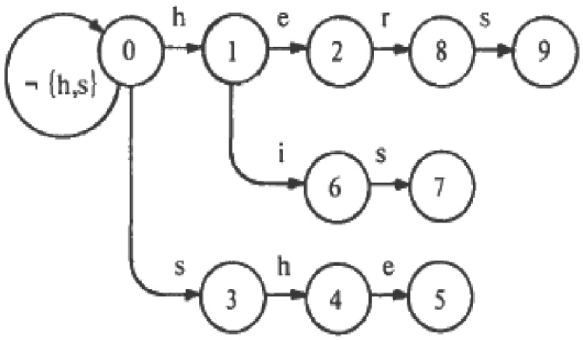

Fig. 1. Aho-Corasick Algorithm

The Boyer-Moore algorithm [4] is also one of the early algorithms and is the most widely used algorithm for string matching. It is based on two heuristics: bad-character heuristic and good suffix heuristic. The novel aspect of the Boyer Moore algorithm and the reason for its effectiveness is that character matching is performed right-to-left. The bad character heuristic shifts the search string to align the mismatching character with the rightmost position at which the mismatching character appears in the search string. If the mismatch occurs in the middle of the search string, then there is suffix that matches. The good suffix heuristic shifts the search string

ISSN: 2277-3754
International Journal of Engineering and Innovative Technology (IJEIT)
Volume 1, Issue 2, February 2012

to the next occurrence of the suffix in the string. For example, given a single pattern of length n to match, we look ahead in the input string by n characters. If the character at this point does not match with the pattern, we move the search pointer ahead by n + 1 characters without inspecting the characters in between. If there is a match, we start comparing the previous characters. The Boyer-Moore algorithm shows a sub linear performance in the average case.

Rafiq et al. simplified the Boyer-Moore algorithm, reduced the memory requirements, and made it faster [15]. In the original Boyer-Moore algorithm, the mismatched character of text is searched in pattern. In their proposed algorithm, the mismatched character of pattern is searched in text. In addition, they made optimizations in calculating jumps between characters and strings. The proposed algorithm's time complexity is O (n/m) in best case, O (n) in average case, and O(nm) in worst case, where n is the length of text and m is the length of pattern.

Parallel String Matching: The first optimal parallel string matching algorithm was proposed by Galil [8].On SIMD-CREW model, this algorithm required n/logn processors, and the time complexity is O (logn); on SIMD-CREW model, it required $n/\log^2 n$ processors and the time complexity is O ($\log^2 n$). Vishkin [9] improved this algorithm to ensure it is still optimal when the alphabet size is not fixed. In [10], an algorithm used $O(n*m)$ processors was presented, and the computation time is O (log log n). A parallel KMP string matching algorithm on distributed memory machine was proposed by CHEN [11].The algorithm is efficient and scalable in the distributed memory environment. Its computation complexity is $O(n/p+m)$,and P is the number of the processors.

Most proposed parallel string matching algorithms are usually based on PRAM (Parallel Random Access Machine) computation model [8]-[10], and this model is not practical in realistic distributed computation environment. CHEN [11] designed a parallel KMP (Knuth-Morris-Pratt) string Matching algorithm [5] on distributed memory machine. The distributed memory model is practical in most parallel machines such as massive parallel processor machine and cluster machine. In PRAM model, all those operations are abstracted as they have the same cost, but in the realistic implementation, it is not the case. Memory access and communication between processors usually spend much more clock cycles than a single computation in processor. So, in the practical design of parallel algorithms, those impact factors should been concerned.

Approximate String Matching: The approximate String Matching problem consisting of k-mismatches and k-differences problems. Since high performance software algorithms are mostly multiphase algorithms which use pre processing with table look up methods, they can not be applied to design special purpose hardware string matcher. Because pure dataflow scheme is suitable to design systolic array architecture, we will show how these dataflow algorithms are used to build special purpose architecture to design string matching hardware.

K-Mismatches problem :With given reference string T (|T| = n) and pattern P (|P| = m), k-mismatches problem consists of finding all approximate occurrences (starting positions) of a pattern string in a reference string with at most k mismatches. Naive dynamic programming method [29] can solve this problem in O (mn) time and O (mn) space. Landau and Vishkin [20] presented O (k (n + m log m)) time and O (k (n + m)) space algorithm. Galil and Giancarlo [13] improved this algorithm with O(kn + m log m) time and O(m) space although it performs worse in practice. Generalized Boyer-Moore algorithm [28] was developed to solve this problem in O (kn(1/(m-k)+(k/c))) expected time where "c" denotes the size of the alphabet. Automata based method with linear time complexity can be found in [3]. Except the naive dynamic programming method, all the algorithms require extra time and space for pre processing that are not included in the complexities shown above. The preprocessing part consists either of gathering useful information about pattern and reference strings or of building the finite state machine.

K-Differences problem: More approaches have been developed for the k-differences problem which is beyond the scope of this paper but, very useful in fields including molecular biology.

II. CONTENT-ADDRESSABLE MEMORY

Traditional computers rely upon a memory design that stores and retrieves data by its address rather than its contents [5]. This retrieval-by-address approach has become very successful due to its simplicity. In late 1970's, the separation between the CPU and memory lead to what is known as the von Neumann bottleneck, where the memory-access path becomes the limiting factor for system performance. Today, the cache hierarchy between CPU and main memory addresses some of the performance issues of the bottleneck. In content-addressable memory (CAM) (also known as associative memory), information is stored, retrieved, or modified based on the data itself, rather than its arbitrary storage location. The basic functions in an associative memory include broadcasting and comparison of a search key with every stored location simultaneously, and identification of the matching words. However, it is difficult to decide where in the memory to write information in such memory architectures. Since data are not addressable by location, methods of identifying currently available memory areas must be employed. For example, free memory areas can be identified by a separate tag bit, or by their contents. CAMs are useful for applications such as data processing applications, database applications, file maintenance, pattern recognition, symbolic representation of information, data retrieval, speech recognition, spell checking, list and string processing, language translations, and networking applications including

ISSN: 2277-3754
International Journal of Engineering and Innovative Technology (IJEIT)
Volume 1, Issue 2, February 2012

routing lookups, packet classification and address translation [5]. The cost of a CAM depends on the storage elements, the cell interconnection, and the amount of logic associated with each element. Therefore, a CAM is more expensive than a random access memory (RAM) because each cell must have storage capability as well as logic circuits for matching its content with an external argument. For this reason, CAMs are just used in applications where the search time is very critical and must be very short.

Furthermore, there are three storage and retrieval architecture types for CAMs: bit serial, byte serial, and word serial. The data in a bit-serial CAM is inspected one bit at time, with the same bit in every word in the memory simultaneously. Byte-serial CAMs inspect a byte with each memory access, comparing the same byte position in every memory word simultaneously. Word-serial systems read one whole word at a time and compare all the words in the memory with each associative access.

III. NETWORK INTRUSION DETECTION SYSTEM

Traditionally networks have been protected using firewalls that monitor and filter network traffic. Firewalls usually ex-amine packet headers to determine whether to block or allow packets. Due to busy network traffic and smart attacking schemes (e.g. Code Red), firewalls are not as effective as they used to be. Signature-based network intrusion detection systems (NIDS) use patterns of well-known attacks to match and identify intrusions. They monitor incoming traffic for suspicious packet contents by inspecting packet payload for attack signatures. Therefore, they have to do string matching in incoming packet contents and often rely on exact string matching approaches. String matching is the most computationally expensive task in intrusion detection systems, and must be performed at wire-speed so that it does not become a bottleneck to the system's performance. It is difficult for software-based packet monitors to keep up with rapidly increasing networking speeds; therefore, software-based solutions are not as effective as hardware-based solutions. Snort [20] is a freely available and widely used intrusion detection tool. It uses a set of rules that are derived from known attacks or other suspicious behaviors. Rules are added to Snort as new vulnerabilities are discovered. Each rule contains a content string, associated rules for its location, and the type of packet it can appear in. Most of the approaches discussed in this section use Snort rules in their implementations.

Mishina et al. presented a string matching algorithm that is suitable for vector processors in 1993 [12]. The algorithm was designed for Hitachi's pipelined vector processor, Integrated Database Processor (IDP). IDP was first commercially introduced in 1986 as an optional hardware for one of Hitachi's general-purpose computers. It was attached to the CPU of a host machine to improve the host machine's performance. The proposed string matching algorithm consists of two parts: cutout and check. In the cutout part, a text string is divided into independently operable substrings so that each substring can be matched with pattern strings in a pipelined manner using vector processors. The cutout part was implemented as an added instruction of the IDP. The idea in the check part is to apply the Aho-Corasick string matching algorithm con-currently to all substrings obtained from the cutout procedure. Experiment results show that the proposed algorithm is 10 times faster than a scalar program using the Aho-Corasick algorithm.

Sidhu et al. proposed an approach based on nondeterministic finite automaton (NFA) for regular expression matching [19]. A regular expression is a pattern that matches one or more strings of characters. An NFA is a directed graph in which each node is a state and each edge is labeled with a single character or an empty string. In the proposed approach, regular expressions are generated for every string in the rule set and an NFA that examines the input one byte at a time is implemented. The time complexity of this approach is O (n), and the area complexity is O (n^2). The time complexity to search through a text of length m is O (n + m), and the area complexity is O(n^2). Experiment results show that the proposed architecture can achieve 0.75 Gbps on a Vertex 100 device with operating frequency of 93.5 MHz, and the area required per search pattern is approximately 31 logic cells. It is important to note that FAs are generally complex, hard to implement, have to be rebuilt every time a string is added and achieve low throughput.

Tuck et al. proposed modifications to the well-known Aho-Corasick string matching algorithm to reduce the amount of memory required to store known malicious strings and improve worst-case timing [22]. They achieve results in both of these areas, while slightly degrading average-case performance. The proposed data storage methods for string matching are bitmap compression and path compression. This compact storage is desired to fit the data structure in on-chip SRAM or the cache of a commodity processor. Their experiments consider both an ASIC and a programmable router design. The ASIC design is tailored to only string matching, while the programmable design assumes an implementation that can be used for many different types of router applications. Experiment results show that the proposed compression optimizations resulted in a 50 times reduction in database size over the Aho-Corasick implementation.

Cho *et al.* proposed a rule-based inspection firewall system based on a parallel architecture [6]. Figure 2 illustrates the pro-posed parallel architecture[2]. Each rule is separately processed in parallel. First, packet data is passed to the units through a 32-bit bus. Then, the header information of each packet is compared with the predefined header data. If there is a match, the payload

ISSN: 2277-3754
International Journal of Engineering and Innovative Technology (IJEIT)
Volume 1, Issue 2, February 2012

data is sent to the content pattern match unit where the predefined pattern is searched. The content pattern match units contain 8-bit registers and 8-bit comparators. To increase throughput, four bytes of data are matched in each stage of the pipeline. In other words, four parallel comparators are used per string. Experiment results show that the proposed architecture can achieve over 2.88 Gbps on an Altera EP20K with operating frequency of 90 MHz, and the area required per search pattern is 10 logic cells.

Sourdis et al. proposed an FPGA-based approach for the string matching problem in network intrusion detection systems [21]. The reason they chose an FPGA-based approach is because of hardware speed and also parallelism can be exploited. Figure 3 illustrates the proposed system[3]. Packets arrive and are distributed to the matching engines. There are N parallel comparators that can process N characters per cycle. The matching results are encoded to determine the action for packets

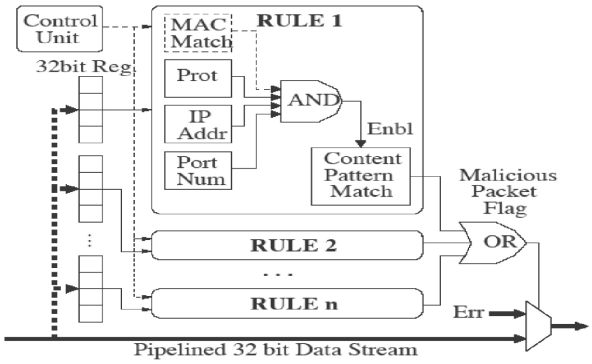

Fig.2.Parallel Data path of NIDS System

The parallel comparators have a CAM-like functionality. The proposed architecture uses fine-grain pipelining for all the modules illustrated in Figure 3. Experiment results show the best throughput that can be achieved is 11 Gbps on a Virtex2 1000 device with operating frequency of 340 MHz, and the match cost per search pattern character is approximately 17 logic cells.

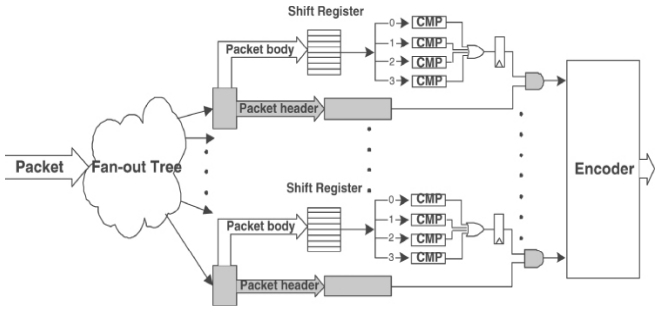

Fig. 3. FPGA-Based NIDS System

Attig et al. proposed Bloom filter architecture for intrusion detection systems, as illustrated in Figure 4. Packets enter the system and are processed by Internet Protocol (IP) wrappers. The data in the packet goes to the input buffer and then flows through the content pipeline. As the packet passes through the pipeline, multiple Bloom engines scan different window lengths for signatures of different lengths. Data leaves the content pipeline, flows to the output buffer, streams through the wrappers, and then packets are re-injected into the network. If a Bloom engine detects a match, a hash table is queried to determine if an exact match occurred. If the queried signature is an exact match, the malicious content can be blocked and an alert message is generated. Experiment results shows that the proposed architecture can achieve a throughput of over 2 Gbps on a Vertex E 2000 device. Figure 2 is courtesy of [6]. Figure 3 is courtesy of [21].

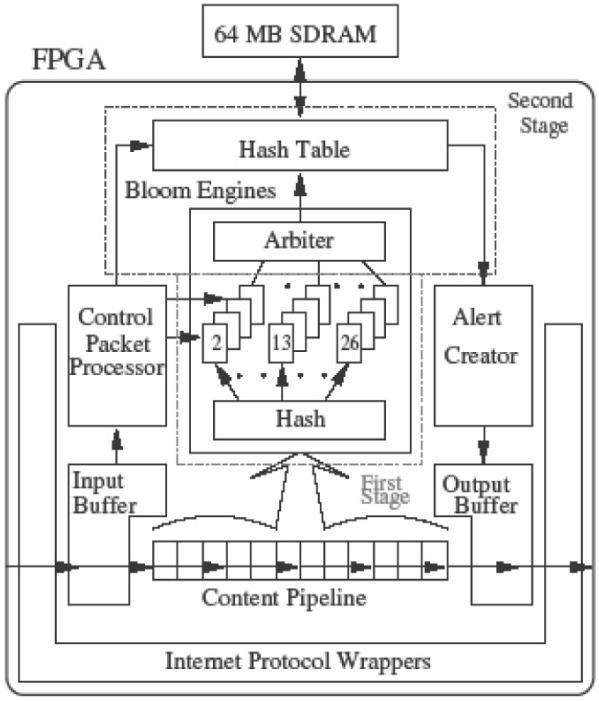

Fig. 4. Bloom Filter Architecture

Rafiq et al. presented systolic array architecture for string searching/matching by using a formal procedural approach [16]. Their approach includes drawing a dependency graph for a simple string searching/matching algorithm, and then forming systolic array architecture from the dependency graph. Figure 5 illustrates systolic array architecture when P is broadcasted and T is pipelined 7. There are $n - m + 1$ processing units in this approach. A subset of T, Tk , is pipelined to the k th processor, where $0 \leq k \leq n - m$. Then, match/mismatch is produced at Tk after m time units in thek th processing unit. The time complexity is O(m) and the area complexity is $O(n - m + 1)$. If $n >> m$, then the area-time complexity is O(nm).

Aldwairi et al. suggested a reconfigurable memory based accelerator for intrusion detection [2]. The accelerator is a part of the configurable network processor architecture shown in Figure 6. It consists of a 2-wide multiple issue VLIW processor with hardware support for eight hyper threads. The memory system consists of multi-port RAM and a high speed DMA.

ISSN: 2277-3754
International Journal of Engineering and Innovative Technology (IJEIT)
Volume 1, Issue 2, February 2012

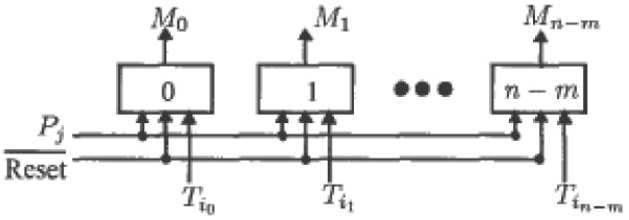

Fig. 5. Systolic Array Architecture (P is broadcasted, T is pipelined)

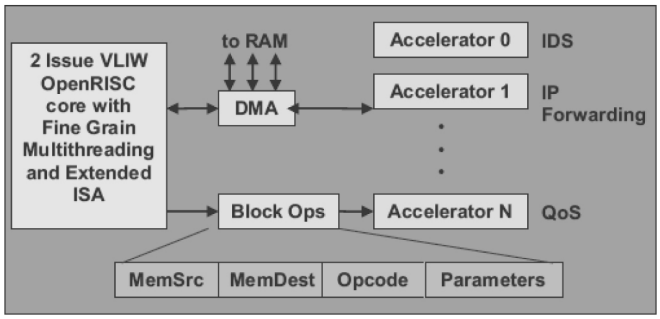

Fig. 6. Configurable Network Processor Architecture

A number of configurable accelerators are used to speed up specific networking tasks such as IP forwarding, quality of service and string matching for intrusion detection. The proposed system consists of two components: software that runs on the VLIW core and a hardware string matching accelerator for FSM implementation. The software generates an FSM and creates the state tables, and the FSM matches strings based on the Aho-Corasick string matching algorithm. They use a different method to store the Aho-Corasick database in an SRAM that achieves a higher throughput than Tuck et al.'s implementation while having a similar memory requirement. It implements a Mealy FSM and consists of a RAM to store the state tables, a register to hold the current state, and control logic to access the RAM and find a match. Experiment results show that the proposed architecture can achieve up to 14 Gbps throughput using 8 FSMs in parallel. The experiments took place on an Altera EP20k400E device. The throughput degrades as the number of characters grows. This is due to the fact that as the number of characters increases, the number of states increases and the state table size increases as well. Figure 4 is courtesy of [3]. Figure 5 is courtesy of [16]. Figure 6 is courtesy of [2].

1) IP A DDRESS L OOKUP IN ROUTERS

IP routing requires that a router perform a longest-prefix match address lookup for each incoming datagram in order to determine the datagram's next hop. Therefore, router performance is limited by the route lookup mechanism. Today, there are two addressing schemes in use: classful addressing scheme and classless inter domain routing (CIDR) scheme. The classful addressing scheme uses a simple two-level hierarchy. The 32-bit IP address is broken down into the network address part and the hosts address part. IP routers forward packets based only on the network address until the packets reach the destined network. Typically, an entry in the forwarding table stores the address prefix (e.g. the network address) and the routing information (i.e. the next-hop router and output interface). The address lookup operation includes finding an exact prefix match in the forwarding table. Arbitrary length prefixes are allowed in the CIDR scheme to provide a more efficient use of the IP address space and avoid the problem of forwarding table explosion. With CIDR, the address lookup operation is more difficult. It includes finding the longest address prefix in the forwarding table that matches the destination address. Several software and hardware solutions to the IP address lookup problem have been published in the literature. In this report, a brief overview of some interesting hardware solutions is presented. The hardware solutions are mainly CAM-based or processor-memory based solutions. In processor-memory based solutions, the routing table is stored in memory and the lookup algorithm runs on a processor. As there may be several memory accesses to retrieve the longest matching prefix, memory bandwidth becomes a bottleneck. Therefore, the objective of many hardware-based IP lookup algorithms is to keep the memory accesses as few as possible. The most straightforward way to implement a lookup scheme is to have a forwarding table in which an entry is designated for each 32-bit IP address. In this case, the size of the forwarding table (next-hop array (NHA)) becomes too large (4 GB) to be practical. Figure 10 illustrates this direct lookup approach 10 . Huang et al. reduced the size of the associated NHA by considering the distribution of the prefixes belonging to the same segment [10]. With 45,000 routing prefixes, it results in about 750 KB entries in the NHA. By employing the concept of compression, the required memory size can be further reduced, but the number of memory accesses increases to three. The time complexity for building the so-called Code Word Array (CWA) and the compressed NHA (CNHA) is O(n log n), where n denotes the number of prefixes in a segment [10]. However, due to the mechanism of compression, it is needed to rebuild the CNHA for updating the forwarding table. Since the routing updates may occur every few seconds, the performance might degrade severely due to the memory bandwidth contention.

Waldvogel et al. proposed an address lookup scheme based on binary search of hash tables [23]. The scheme requires a worst-case time of log2 (address bits) hash lookups and additional memory space to store markers remembering the last found best matching prefix (BMP). Adding or deleting a single prefix can change the BMP values of a large number of markers, and hence updating the forwarding table is expensive in the scheme. Waldvogel et al. presented a binary search algorithm in each prefix length using hashing, but the binary search takes times of memory accesses in worst case which is the number of distinct prefixes in IP address. The scheme

ISSN: 2277-3754
International Journal of Engineering and Innovative Technology (IJEIT)
Volume 1, Issue 2, February 2012

assumes to use a perfect hashing hardware and does not consider the occasion of collisions in hashing.

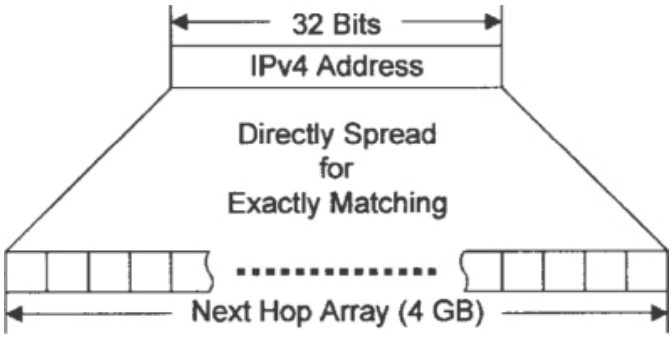

Fig. 7. Direct Lookup Approach

Gupta et al. presented a route lookup mechanism that when implemented in a pipelined fashion in hardware, can achieve one route lookup every memory access [9]. Figure 7 is courtesy of [10].Figure 8 illustrates the hardware scheme presented [8] . The approach employs a multi-level lookup mechanism, and uses two tables, both stored in DRAM. The first table stores all possible route prefixes that are up to 24 bits long (i.e. 224 entries). If an entry is longer than 24 bits, the first table entry contains a pointer to a set of entries in the second table. One observation in this approach is that in a typical router, most of the entries have prefixes of length 24 bits or less. As a result, using 24 bits in the first table allows the match to be found in one memory access for the majority of cases. This approach, in general, requires two memory accesses for a lookup in a 33 MB forwarding table, and achieves up to 20 million lookups per second. By using an additional intermediate length table and splitting the 32-bit address into three portions of 21, 3, and 8 bits, respectively, the amount of memory can be reduced to 9 MB. However, it uses memory inefficiently; insertion and deletion of routes from the tables may require many memory accesses.

Pao et al. presented a hardware architecture modeled as a searching problem on a binary trie [14]. A trie is an ordered tree data structure that is used to store an associative array where the keys are strings. Unlike a binary search tree, no node in the tree stores the key associated with that node; instead, its position in the trie shows what key it is associated with. In Pao et al.'s approach, the complete binary-tree is partitioned into non-overlapping sub trees of 255 nodes. Each sub tree is represented using a bit-vector and can be searched in parallel. A k-bit binary-trie can be represented by a bit-vector with $2k+1 - 1$ bits, called the tree-vector. The bits in the tree-vector are numbered from left to right. Bit i of the tree-vector is equal to 1, if the prefix that corresponds to node i of the binary-trie is present in the forwarding table, otherwise bit i is equal to 0.Similarly, the search path can be represented by a bit-vector called the mask-vector. To find the best matching prefix in a sub tree, the following operations are performed in a pipelined architecture:

(i) Read the tree-vector and the mask-vector from memory

(ii) Perform a bit-wise AND operation of the mask-vector and the tree-vector to obtain the result-vector and find the position of the rightmost '1' in the result-vector

(iii) Read the next-hop identifier from the routing-vector The longest matching prefix, if any, is given by the bit number of the rightmost '1' in the result-vector of the AND operation. To reduce the processing time of step (ii), a digital circuit called the RMB locator circuit is implemented. If all data structures are stored in SRAM, a cycle time of 12.5 ns is achieved. If DRAM is preferred, then the cycle time is about 50 ns. The total amount of memory required for a reasonable implementation is about 8 MB. Together with a pipelined architecture, a throughput of 20 millions and 80 millions lookup per second can be achieved if the major memory modules are implemented using DRAM and SRAM, respectively.

Mohammadi et al. state that software lookups are flexible and scalable but slow, and hardware lookups are fast but inflexible [13]. Therefore, they propose hardware assisted software lookups by making small modifications in the instruction set of a general-purpose processor. Figure 8 is courtesy of [17]The approach is based on DMP Tree (Dynamic M-way Prefix Tree), which is a superset of the B-Tree data structure. In general, a B-Tree consists of several buckets, where each bucket is a node, and contains sorted elements and pointers to the next level. To find the longest prefix in the DMP-Tree, the following instructions are implemented in hardware to help software lookups:

[1] prefix matching

(ii) Finding place of an IP address in a bucket

(iii) Finding the longest matching prefix of an IP address in bucket.

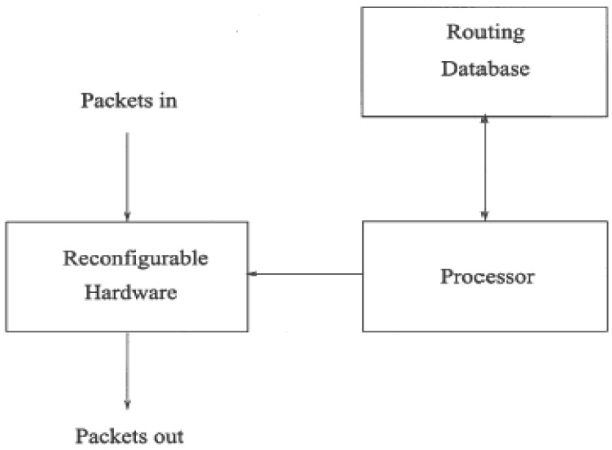

Fig. 8.Multi-level Lookup Approach

Experiment results show that the number of lookups decreases as the branching factor increases. For instance, when the branching factor is 4, 12 million lookups per second can be achieved in a routing table of 100 KB. However, when the branching factor is 40, 4 million lookups per second can be achieved.

ISSN: 2277-3754
International Journal of Engineering and Innovative Technology (IJEIT)
Volume 1, Issue 2, February 2012

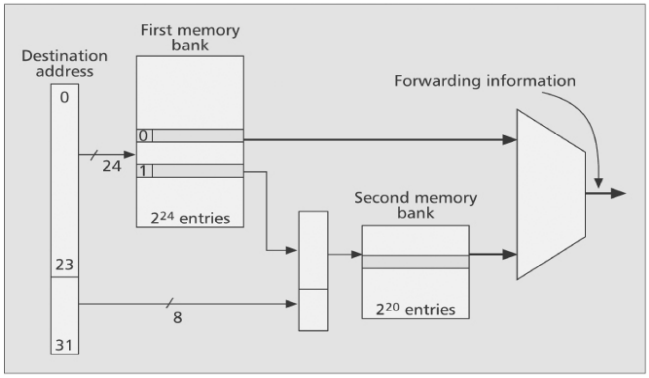

Fig. 9. Lookup Engine Architecture

Sangireddy et al. suggest that most of the available schemes for IP address lookup depend on the memory access technology which limits their performance [18]. Therefore, they propose a binary decision diagram (BDD) based computational logic for address lookup scheme. One key observation in this scheme is that even at the largest network access point; the number of next-hop ports (NHPs) is generally not greater than 256. Hence, an NHP associated with any prefix in the routing table can be encoded using an 8-bit binary code. Therefore, the computational logic design is done for eight output bits, and that gives eight BDDs to be processed. The results show that the BDD hardware engine gives a throughput of up to 175.7 million lookups per second for a large AADS routing table with 33,796 prefixes, a throughput of up to 168.6 million lookups per second for an MAEWest routing table with 29,487 prefixes, and a throughput of up to 229.3 million lookups per second for the PacBell routing table with 6,822 prefixes. Different approach from CAM-based and processor memory based solutions was proposed by Desai et al. [7]. They present architecture for an IP address lookup engine based on programmable finite-state machines (FSMs). Basically, the lookup engine is a reconfigurable hardware implementation in the form of a FSM. Figure 9 illustrates the architecture of the lookup engine [9]. The reconfigurable hardware illustrated in Figure 9 performs the address lookup. The processor computes the FSM for a given routing database of address prefixes and then compiles it in a format appropriate for programming the reconfigurable hardware. If there is a change in the routing database, then the state machine is recomputed again. Experiment results show that the average number of cycles required for lookup is about 5 to 8 cycles [7].

IV. DISTRIBUTED ARCHITECTURE AND ALGORITHM

A. The Distributed String Matching Algorithm Architecture: Nowadays, the frequency of the processor is high. The speed gap between processors and memory access and communications between processors is large. Because the communication between different computing nodes requires the message to be sent by operating system and the network protocol is heavy for a single communication. It usually takes much more cycles to send data to other computing nodes than computing it in local processor. Our distributed string matching algorithm architecture is based on the following three assumptions:

1) The computing environment is distributed memory environment, such as cluster computing environment.

2) Communication between processors costs much more time than computing it in local processors.

3) The length of text is much longer than the length of the pattern string. The text is partitioned and then assigned to each processor before processing. The proposed distributed architecture is shown in Fig 10.

In step 1, the processor with number zero broadcasts the pattern string to other processors. The length of the pattern is ***m***, and stores in an array named pat. This step can be implemented by the binomial tree-based communication strategy or the Fibonacci communication strategy [15].In step 2, all the processors call the procedure "BUILD" to Figure 9 is courtesy of [7] Figure 10 is courtesy of [32].preprocess the pattern string in parallel. For example, if you want to parallel BM algorithm, you need to compute the "bad character" shift and "good suffix" shift in this step. In step 3, all the processors call the string matching algorithm to search the pattern string locally in the text ***Ti***. ***Ti*** denotes the text assigned to processor whose number is ***i***. The string matching algorithm could be all the classical string matching algorithms, such as the KMP algorithm and the BM algorithm. In step 4, the boundary condition should be considered, because the pattern matching may occur just across the text ***Ti*** and ***Ti+1***. The boundary pattern matching condition can be solved by sending the last ***m-1*** characters' information from processor ***PEi*** to the next processor ***PEi+1***.

```
(1) PE0 broadcast pattern string pat [1, m]
(2) for i=0 to p-1 par-do
        Call procedure BUILD
    end for
(3) for i=0 to p-1 par-do
        StringMatchingAlgorithm (Ti, pat)
    end for
(4) Deal with the boundary condition
```

Fig.10. the distributed string matching algorithm architecture

B. The Improved Single String Matching Algorithm: Our improved single string matching algorithm is based on the algorithm Fast-Search published by Cantone and Faro [7]. The aim of our algorithm is to improve the worst case complexity of the Fast-Search algorithm while keeping its practical efficiency. The key idea of our algorithm is that when a pattern matching occurs, the position in the text is recorded, and in the next matching attempt, a pattern matching will be reported when the

ISSN: 2277-3754
International Journal of Engineering and Innovative Technology (IJEIT)
Volume 1, Issue 2, February 2012

character comparison comes to that position instead of the end of this attempt. This method was first discovered by Galil [13].In this algorithm, we have not presented the definition and the construction algorithm for arrays named "bmBc" and "bmGs", because the definition and computation algorithm can be found in the web site [12] through the hyperlink titled"Boyer-Moore algorithm". We use the variable "end" to record the position, and the variable "pos" indicates where the current matching attempt should stop and a matching position will be reported at that time. In section IV, we will prove that the worst-case time complexity of this improved algorithm is $O(n + m)$.

V. IMPLEMENTATION OF STRING MATCHING ON MULTIPROCESSOR USING DIVIDE AND CONQUER TECHNIQUE

S.Viswanadha Raju et al. proposed a string matching on multiple processors using divide and conquer technique [31].Let T be the size of the text, P be the number of processors, m be the size of the pattern, r be the number of sub texts. We decompose T into r subtexts, where each subtext contains (T/P) +m-1 successive characters of T. There is an overlap of m-1 string characters between successive subtexts, that is, a redundancy of r(m-1) characters[6]. Alternatively it could be assumed that the database of an information retrieval system contains r independent documents. Therefore, all the above partitions yield a number of independent tasks each comprising some data (i.e. a string and a large subtext) and a sequential string matching procedure that operates on that data. The main issue to be addressed is how the several tasks (or r subtexts) can be mapped or distributed to multiple processors for concurrent execution. In two different ways we can distribute the database across a multi computer networks. If r=P then each r contains T/P+m-1 characters. This is called static allocation of subtext.

A .Distribution of text to different processors:

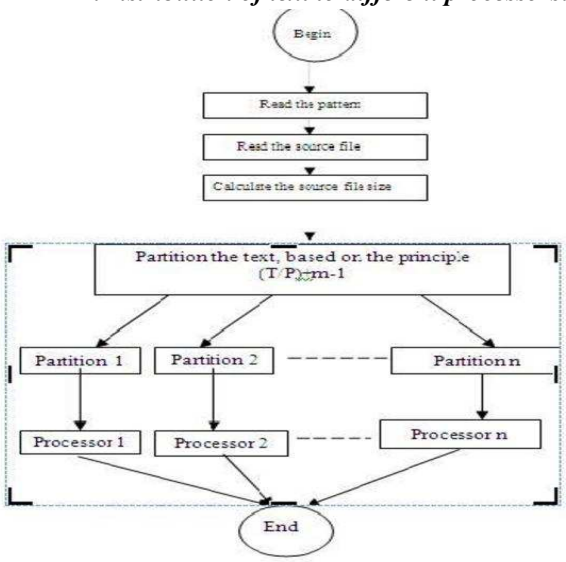

Fig.11.Distribution of TEXT to different processors

From the Figure 11, Consider the pattern file and text file. Then calculate the sizes of the files. Read the pattern and text files, Partition the text file based on Divide and Conquer Technique with overlapping. Text is divided into multiple sub texts depends on the number of processors and pattern size. These subtexts are assigned to threads. Check LAN connection before distributing data to different processors. Then assign each thread to different processors. Search using Brute-Force algorithm done on each processor individually. Finally, it returns all matched locations and time to search

VI. EXPERIMENTAL RESULTS

The preceding section discussed how to implement the searching on multiprocessors. This section discusses Experimental Results, which represented in the form of tables based on the previous section. From that representation we present graphs, which are drawn between P processors where 0<P<5 and time, by taking K patterns where 0<K<4 as key factor. Generally the performance of the algorithm based on the number of operations, size of the text and pattern and also on the number of processors. Not only on these operations but also it depends on the number of occurrences of pattern considered. Figure 11 is courtesy of [34].Figure 12 is courtesy of [32].which is most important for improving the performance of the algorithm. Java is used to implement this work using Windows XP platform. We have considering three files for the implementation discussed in the previous chapter such as f_1 of size 1.43 Mb, f_2 of size 2 Mb, and f_3 of size 3.20 Mb. The pattern files are p_1, p_2, p_3 with respect to those three files. Here bytes mean number of characters. Time is measured in milli seconds

$p_{i,j,}$ represents pattern i in file j

Ex : $p_{1,1}$ gives pattern 1 in file 1 (f_1)

$p_{1,2}$ gives pattern 1 in file 2 (f_2)

File 1 The pattern files that are searched in the text file f_1 are $p_{1,1}$ of size 6 bytes, $p_{2,1}$ of size 4 bytes, and $p_{3,1}$ of size 7 bytes has to be found using Brute-Force algorithm.

File 2 The pattern files that are searched in the text file f_2 are $p_{1,2}$ of size 8 bytes, $p_{2,2}$ of size 4 bytes, and $p_{3,2}$ of size 6 bytes has to be found using Brute-Force algorithm.

File 3 The pattern files that are searched in the text file f_3 are $p_{1,3}$ of size 3 bytes, $p_{2,3}$ of size 7 bytes, and $p_{3,3}$ of size 9 bytes has to be found using Brute-Force algorithm.

VII. OTHER HARDWARE APPROACHES

Because speed is the essential factor of those applications that need string matching tasks, special purpose hardware should be attached to the host computer as a peripheral device like sorter or FFT device. Figure-12 shows the organization of general purpose computer with such special purpose devices attached. Some serial and automata based hardware algorithms and implementations for the exact matching problem are proposed in the literature. Mukhopadhyay [25] proposed a primitive nonnumeric processor scheme in which the

ISSN: 2277-3754
International Journal of Engineering and Innovative Technology (IJEIT)
Volume 1, Issue 2, February 2012

characters of pattern string are preloaded in each processing element and input characters of the reference string are applied (broadcast) to all processing elements. Foster and Kung [11] proposed design of VLSI chip for which linear systolic array of cells (processing elements) are used.

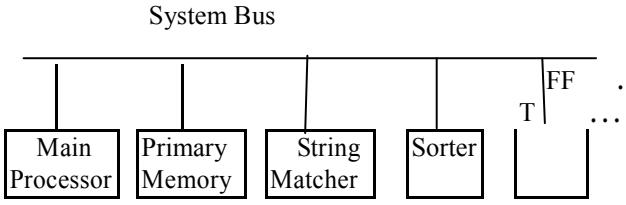

Fig.12.System block diagram of general purpose computer

This scheme uses alternative input of pattern and reference string characters one at a time into the array of cells from both directions accordingly. During each pair of consecutive time slices the chip inputs two characters and returns one output. Thus the required time slices to process entire reference string of length n is2n which is twice its length. To minimize the number of cells to the number of characters in the pattern, the pattern recirculation method (i.e. the last character of the pattern followed by the first character of the pattern again) can be used in this approach. Architectural design scheme and the simulation of the automata based approach are found in [17]. Since this scheme needs to construct the finite state machine, complex control communications and memory spaces to manage the automata action are required.

VIII. DATAFLOW SCHEME: IMPLICIT PARALLELISM

Design approach: Dataflow machines have been proposed and developed steadily as an attractive instruction level parallel all its required operands are available based on data driven mechanism and any set of enabled operations can be executed in parallel. To be a good parallel machine, it should also provide good performance on non-computational tasks such as string matching problems. In dataflow environment, array or table handling is a critical problem [15, 21] and we should not use multi-phase table look up methods that most high performance string matching algorithms use. They are also not proper for the special purpose hardware string matcher design since it needs elegant systolic algorithm. Thus we need high performance dataflow scheme using only data flow without preprocessing and table look up methods. This pure dataflow scheme can be used to build VLSI chip since it does not use any memory references and can be converted to systolic array of cells. For matching each pattern character with each reference character, we use the method of storing pattern string in the array of cells and processing the reference string from one direction into the array of cells [24, 25].As Foster and Kung [11] mentioned, the good algorithms for VLSI implementation are not necessarily those requiring minimal computation. Computation is cheap in VLSI and the communication determines the performance.

IX. CONCLUSION

In this report, we have presented several hardware approaches for exact string matching. Network intrusion detection (NIDS) and IP address lookup in routers are two applications that have been discussed for string matching. The approaches presented for NIDS applications include CAM based solutions, finite automata, discrete comparators, bloom filters, and systolic array structures, while the approaches presented for IP lookup include CAM-based solutions, multi-level lookup, binary search, binary tries, hashing, binary decision diagrams, and finite state machines, Distributed architectures with dataflow schemes, and multi-processors using divide and conquer techniques.

REFERENCES

[1] A. V. Aho and M. J. Corasick. Efficient String Matching: An Aid to Bibliographic Search. Communications of the ACM, 18(6):333–340, 1975.

[2] M. Aldwairi, T. Conte, and P. Franzon. Configurable String Matching Hardware for Speeding up Intrusion Detection. ACM SIGARCH Computer Architecture News, 33(1):99–107, 2005.

[3] M. Attig, S. Dharmapurikar, and J. Lockwood. Implementation Results of Bloom Filters for String Matching. In Proceedings of the 12th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM'04), Napa, California, USA, April 2004.

[4] R. S. Boyer and J. S. Moore. A Fast String Searching Algorithm. Communications of the ACM, 20(10):762–772, 1977.

[5] L. Chisvin and R. J. Duckworth. Content-Addressable and Associative Memory: Alternatives to the Ubiquitous RAM. IEEE Computer, 22(7):51–64, 1989.

[6] Y. H. Cho, S. Navab, and W. H. Mangione-Smith. Specialized Hardware for Deep Network Packet Filtering. In Proceedings of the 12th International Conference on Field-Programmable Logic and Applications (FPL'02), pages 452–461, London, UK, 2002. Springer-Verlag.

[7] M. Desai, R. Gupta, A. Karandikar, K. Saxena, and V. Samant. Reconfigurable Finite-Sate Machine Based IP Lookup Engine for High-Speed Router. IEEE Journal on Selected Areas in Communications, 21(4), May 2003.

[8] S. Dharmapurikar, P. Krishnamurthy, T. S. Sproull, and J. W. Lockwood. Deep Packet Inspection using Parallel Bloom Filters. IEEE Micro, 24(1):52–61, 2004.

[9] P. Gupta, S. Lin, and N. McKeown. Routing Lookups in Hardware at Memory Access Speeds. In Proceedings of IEEE Infocom'98, volume 3, pages 1240–1247, San Francisco, USA, March 1998.

[10] N. Huang and S. Zhao. A Novel IP-Routing Lookup Scheme and Hardware Architecture for Multigigabit Switching Routers. IEEE Journal on Selected Areas in Communications, 17(6), June 1999.

ISSN: 2277-3754
International Journal of Engineering and Innovative Technology (IJEIT)
Volume 1, Issue 2, February 2012

[11] H. Lim and J. Seo andY. Jung. High Speed IP Address Lookup Architecture Using Hashing. IEEE Communications Letters, 7(10):502– 504, October 2003.

[12] Y. Mishina and K. Kojima. String Matching on IDP: A String Matching Algorithm for Vector Processors and its Implementation. In Proceedings of 1993 IEEE International Conference on Computer Design (ICCD'93), 1993.

[13] H. Mohammadi, N. Yazdani, B. Robatmili, and M. Nourani. HASIL: Hardware Assisted Software-based IP Lookup for Large Routing Tables. In Proceedings of the 11th IEEE International Conference on Networks (ICON2003), pages 99–104, September 2003.

[14] D. Pao, C. Liu, A. Wu, L. Yeung, and K. Chan. Efficient Hardware Architecture for Fast IP Address Lookup. In Proceedings of IEEE Infocom 2002, New York, USA, June 2002.

[15] A. N. M. E. Rafiq, M. W. El-Kharashi, and F. Gebali. A Fast String Search Algorithm for Computer Networking. In Proceedings of the IEEE Pacific Rim Conference on Communications, Computers and Signal Processing, volume 2, pages 764–767, August 2003.

[16] A. N. M. E. Rafiq, F. Gebali, and M. W. El-Kharashi. A Systolic Array Structure for String Searching. In Proceedings of the International Conference on Electrical, Electronic and Computer Engineering, (ICEEC'04), pages 281–284, September 2004.

[17] M. A. Ruiz-Sanchez, E. W. Biersack, and W. Dabbous. Survey and Taxonomy of IP Address Lookup Algorithms. IEEE Network, 15(2):8– 23, April 2001.

[18] R. Sangireddy and A. K. Somani. High-Speed IP Routing with Binary Decision Diagrams Based Hardware Address Lookup Engine. IEEE Journal on Selected Areas in Communications, 21(4), May 2003.

[19] R. Sidhu and V. K. Prasanna. Fast Regular Expression Matching Using FPGAs. In Proceedings of the 9th Annual IEEE Symposium on FieldProgrammable Custom Computing Machines (FCCM'01), pages 227– 238, Rohnert Park, CA, USA, May 2001.

[20] Snort. www.snort.org.

[21] I. Sourdis and D. Pnevmatikatos. Fast, Large-Scale String Match for a 10 Gbps FPGA-based Network Intrusion Detection System. In Proceedings of the 13th International Conference on Field Programmable Logic and Applications (FPL'03), Lisbon, Portugal, September 2003.

[22] N. Tuck, T. Sherwood, B. Calder, and G. Varghese. Deterministic Memory-Efficient String Matching Algorithms for Intrusion Detection. In Proceedings of IEEE Infocom, Hong Kong, March 2004.

[23] M. Waldvogel, G. Varghese, J. Turner, and B. Plattner. Scalable High Speed IP Routing Lookups. In SIGCOMM '97: Proceedings of the ACM SIGCOMM '97 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication, pages 25– 36, New York, NY, USA, 1997. ACM Press.

[24] Arvind, and David E. Culler, "Dataflow Architectures," Annual Review in Computer Science, Vol. 1, pp. 225-253, 1986.

[25] M. J. Foster and H. T. Kung, "The Design of Special-Purpose VLSI Chips," Computer, pp. 26-38, Jan. 1980.

[26] D. D. Gajski, D. A. Padua, D. J. Kuck, and R. H. Kuhn, "A Second Opinion on Data Flow Machines and Languages," Computer, Vol. 15, pp. 58-69, Feb. 1982.

[27] Z. Galil and R. Giancarlo, "Improved String Matching with K Mismatches," SIGACT News 17, pp. 52-54, 1986.

[28] Zvi Galil, "A Constant-Time Optimal Parallel String-Matching Algorithm," Journal of the ACM, Vol. 42, pp.908-918, July 1995.

[29] Merrill E. Isenman and Dennis E. Shasha, "Performance and Architectural Issues for String Matching," IEEE Trans. on Computers, Vol. 39, No. 2, pp. 238-250, Feb. 1990.

[30] D. E. Knuth, J. Morris, and V. Pratt, "Fast Patern Matching in Strings," SIAM J. of Comput., Vol. 6, pp. 323-350, 1977.

[31] G. Landau and U. Vishkin, "Efficient String Matching with K Mismatches," Theoretical Comput. Sci., 43, pp. 239-249, 1986.

[32] Bi Kun ,Gu Nai-jie,Tu Kun,Liu Xiao-Hu,and Liu Gang "A Practical Distributed String matching Algoritm Architecture and Implementation WASET vol .19,pp.156-162,oct.2005.

[33] Zheng liu,Xin chen,james Bornman and Rao Jiang ,"A fast algorithm for approximate string Matching on gene sequences," in Symposium.16 th annu.combinatiorial pattren matching ,LNCS, Spinger-verlag,Vol.3537,pp.79-90,june.2005

[34] S.Viswanadha Raju,K.S.M.V.kumar," Implementation of String matching on Multiprocessors using divide and Conquer Technique " in 2011 3rd international Conference on Machine Learning and Computing .(ICMLC),WWW.Elsevier.com, Dec 2011.

AUTHOR BIOGRAPHY

K S M V Kumar , Assoc. prof in Department of CSE, Guru Nanak Institute of technology ,Hyderabad, INDIA ,6 No of publications in national and international journals, Memberships :ACM life member. CSI life member.

Prof. S Viswanadha Raju, Professor of CSE, SIT, JNTUH, Hyderabad, INDIA. He has a more than 15 years of experience in Teaching, Research and Administration Presently, Professor in CSE, School of Information Technology, JNT University Hyderabad . **PhD** awarded from the Dept of Computer Science and Engineering, Acharya Nagarjuna University in the area of Information Retrieval. **M.Tech,** First class with Distinction from the Dept of Computer Science and Engineering, Jawaharlal Nehru Technological University, Hyderabad Head, Dept of Computer Science and Engineering, JNTUH College of Engineering, (JNT University Hyderabad) Karimnager from Sep 2010 - Dec 2010. Director, MCA (Accredited by NBA), Gokaraju Rangaraju Institute of Engineering & Technology (GRIET) Hyderabad. From July 2009 to July 2010. Head, Dept of MCA, GRIET, Hyderabad from June 2002 to April 2008.Convener, International Conference on Advanced Computing Technologies (ICACT 2008) at GRIET, Hyderabad His research area includes Information Retrieval, Data Base Management Systems, Data Mining, Analysis of Algorithms, Parallel Programming and related area

ISSN: 2277-3754
International Journal of Engineering and Innovative Technology (IJEIT)
Volume 1, Issue 2, February 2012

He has 20 research publications to his credit Presently 12 Ph.D Scholars in Computer Science and Engineering are pursuing research under his guidance He has life membership in different Professional bodies such as IETE, ISTE, and CSI. **Founder Member of International Journals:** International Journal of Advanced Computing, ISSN: 0975 7686International Journal of Data Engineering and Computer Science ISSN: 0975 8372**Books:** Information Retrieval Systems, Dr.A.Vinaya Babu an Education Book Publishing by Pearsoned Education, Chennai (In printing).

Dr.A.Govardhan did his Intermediate from APRJC Nagarjuna Sagar, during 1986-1988, BE in Computer Science and Engineering from Osmania University College of Engineering, Hyderabad in 1992, M.Tech from Jawaharlal Nehru University(JNU), Delhi in 1994 and he earned his Ph.D from Jawaharlal Nehru Technological University, Hyderabad in 2003.Professor in CSE & Director of Evaluation and Excellence for Outstanding services, Achievements, Contributions, Meritorious Services, Outstanding Performance and Remarkable Role in the field of Education and Service to the Nation He is a Member in Executive Council, JNTUH, He is a member of Standing Committee for Academic Senate, JNT University Hyderabad and Academic Advisory Committee (AAC), UGC-Academic Staff College and Sports Council, JNT University Hyderabad .He is a Member on the Editorial Boards for Seven International Journals. He has chaired 5 sessions at International and 2 sessions at National Conferences. He was a Coordinator for a Research Project. He had been to Stockholm, Sweden for an International Conference. He has organized 10 Workshops and 1 Refresher Course. He guided 7 Ph.D Thesis, 1 M.Phil and 123 M.Tech projects. He has 152 Research Publications in International/National Journals and Conferences. He a has 152 Research Publications in International/National Journals and Conferences. He has delivered more than 35 Keynote addresses and invited lectures on various topics including Research Methodologies in Computer Science & Engineering, Data Warehousing and Mining, Beauty of the Software, WWW & Internet, Research: A Career, Research: A Journey and How to be a Good Teacher. He is also a member in various Professional and Service-Oriented bodies