Parallel String Matching Problems with Computing Models – An Analysis of the Most Recent Studies

K Butchi Raju
Associate Professor, Department of CSE, GRIET, Hyderabad, AP, INDIA

Chinta Someswara Rao
Assistant Professor, Dept of CSE, SRKR Engineering College, Bhimavaram, AP, INDIA-534204,

S. Viswanadha Raju, PhD
Professor & HOD, Department of CSE, JNTU College of Engineering, Jagithyala, A.P., INDIA

ABSTRACT
We survey the current techniques to handle with the problem of parallel string matching with computing models. This is becoming a more and more relevant issue for many fast growing areas such as information retrieval and computational biology. We focus on current developments of parallel string matching, computing models, and the central ideas of the algorithms and their complexities. We present the performance of the different algorithms and their effectiveness. Finally this analysis helps the researchers to develop the better technique.

Keywords- Text processing, IRS, computing models, string matching, parallel algorithms.

1. INTRODUCTION
String matching has been extensively studied in the past 30 years. A string C of length n is a sequence of characters C1C2…Cn. Let Σ={Y1, Y2, … YN} be a finite set of strings called patterns, and let I be an arbitrary string. The string matching problem is to identify and locate all substrings of I which are patterns in Σ. It performs important tasks in many applications including information retrieval; library systems, artificial intelligence, pattern recognition, molecular biology, and text search and edit systems. The challenge is that for the string matching to be accurate, it needs to be able to search every byte of every input data streaming for a potential match from a large set of strings. So, normal software solutions are not enough, for this we need the hardware computing models [1-7].

The main contributions of this work are summarized as follows. This work offers a comprehensive study as well as the results of typical parallel string matching algorithms at various aspects and their application on computing models. This work suggests the most efficient algorithmic models and demonstrates the performance gain for both synthetic and real data. The rest of this work is organized as, review typical algorithms, algorithmic models and finally conclude the study.

2. PARALLEL COMPUTING MODELS
Parallelism takes many forms and appears in many areas. It is exhibited at the CPU level when microinstructions are executed simultaneously. It is also present when an arithmetic or logic operation is realized by a circuit of small depth, as with carry-save addition. And it is present when multiple computers are connected together in a network. Parallelism can be available but go unused, either because an application was not designed to exploit parallelism or because a problem is inherently serial [8-11].

A parallel computer is any computer that can perform more than one operation at time. By this dentition almost every computer is a parallel computer. For example, in the pursuit of speed, computer architects regularly perform multiple operations in each CPU cycle: they execute several microinstructions per cycle and overlap input and output operations (I/O) with arithmetic and logical operations. Architects also design parallel computers that are either several CPU and memory units attached to a common bus or a collection of computers connected together via a network. Clearly parallelism is common in computer science today [8-11].

A parallel programming model is a concept that enables the expression of parallel programs which can be compiled and executed. The value of a programming model is usually judged on its generality: how well a range of different problems can be expressed and how well they execute on a range of different architectures. The implementation of a programming model can take several forms such as libraries invoked from traditional sequential languages, language extensions, or complete new execution models [8-11].

In this, paper we examine a number of explicitly parallel models of computation for string matching, including shared and distributed memory models proposed by different researchers.

3. OUR CONTRIBUTION
Hundreds of articles, literally, have been published about string matching with computing models, exploring the multitude of theoretical and practical facets of this fascinating fundamental problem. For an n-character text T and an m-character pattern x, the classical algorithm by Knuth, Morris and Pratt takes O(n+m) time and uses O(m) auxiliary space to find all pattern occurrences in the text, namely, all text positions i, such that x = T[i … i + m-1]. Many other algorithms have been published; some are faster on the average, use only constant auxiliary space, operate in real-time, or have other interesting benefits. This work categorizes the algorithms into some categories to emphasize the data structure that drives the matching. These categories are discussed here.

3.1. Intrusion Detection Systems (IDS)
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. Here, we discuss some of the techniques.

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 [12]. 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. 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 [13]. Here 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 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.

Sourdis et al. proposed an FPGA-based approach for the string matching problem in network intrusion detection systems [14]. The reason they chose an FPGA-based approach is because of hardware speed and also parallelism can be exploited. Figure 3 illustrates the proposed system3. 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.

Dharmapurikar et al.[15] proposed a hardware architecture based on parallel Bloom filters for network packet inspection. A Bloom filter is a space-efficient probabilistic data structure that is used to test whether or not an element is a member of a set. It stores a set of signatures compactly by computing multiple hash functions on each member of the set. The answer to querying a database of strings to check for the membership of a particular string can be "false positive", but never "false negative". False positive means a condition exists when in fact it does not. False negative means a condition does not exist when in fact it does. The computation time involved in performing the query is independent of the number of strings in the database, provided the memory used by the data structure scales linearly with the number of strings stored in it.

J.Nandhini et al.,[16] provide a systematic virus detection software solution for network security for computer systems. Instead of placing entire matching patterns on a chip, proposed solution is based on an antivirus processor that works as much of the filtering information as possible onto a reference memory. The infrequently accessing off-reference data to make the matching mechanism scalable to large pattern sets. Dual port BITCAM processing program is used along with the Exact Matching Engine and Bloom Filter process. This Dual port BITCAM processes next to the exact matching engine and bloom filter process. This Dual port BITCAM process is placed exclusively for obtaining higher throughput.

Safaa O. Al-Mamory et al.,[17] suggest distributed environment in order to enhance the problems of Snort IDS, one of the problems is the efficiency problem. They achieved this goal by enhancing the Snort's string matching engine through using a LAN of computers, where each computer in the LAN matching a subset of the monitored attacks. Snort is an open source IDS which enable us to detect the previously known intrusion.

Performance evaluation: From fig 1 we clearly observe, that it is possible to improve Snort's efficiency using distributed environment and testability of Snort has been enhanced. At the same time from the fig 2 we say that number of PC's are increased automatically time decreases.

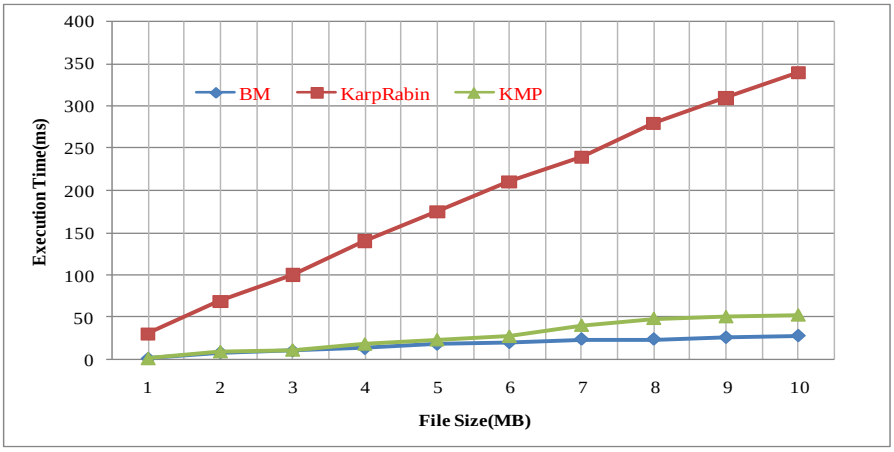

Fig 1 File size VS Execution time

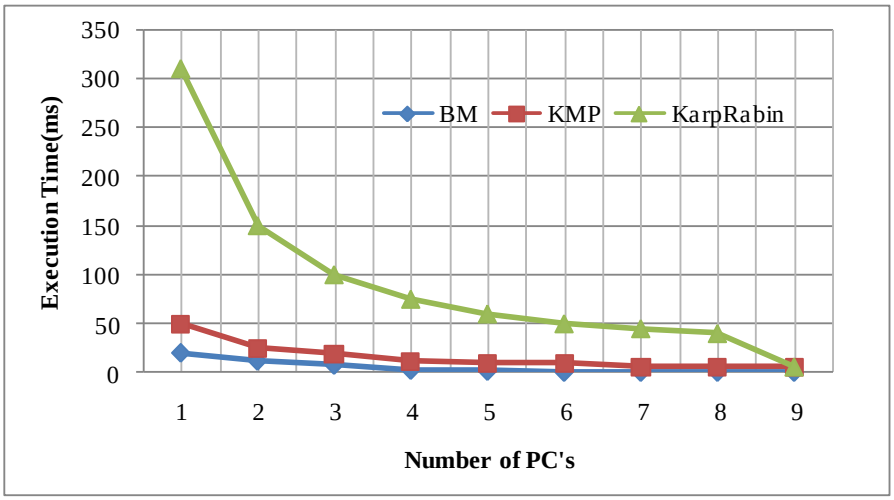

Fig 2 Number of PC's VS Execution time

3.1.1. DFA BASED APPROACHES

Issues in accelerating DFA-based multi-pattern matching have received much attention in recent years by several researchers. Here we discuss some of them.

Dharmapurikar et al. presented a scheme[18] that can process multiple characters per clock cycle and attain average throughput up to multi-gigabit with moderate memory consumption. But in the worst case they must access the relatively slow off-chip SRAMs frequently for exact string comparisons. Nan et al [19]. Introduced a variable-stride method to deal with string matching ruleset without inciting the byte alignment problem. While enhancing the throughput, this method is sensitive to both ruleset and input string causing greatly reduced throughput in worst cases. Lu et al. [20] introduced parallel DFAs with overlapping input windows to achieve the goal of processing multiple characters in each clock cycle. By slight modification to the straightforward representation of the transition rules, the complexity of each DFA is distinctively reduced. Brodie et al.[21] increased the throughput of regex matching by expanding the alphabet set, resulting in an exponentially increased memory requirement in the worst case. A recent method D. Ficara et.al [22] introduced the sampling techniques to accelerate regex matching, but not all kinds of regex are supported.

Seongyong Ahn et.al [23] proposed a hardware-efficient string matching architecture using the brute-force algorithm. A process element that organizes the proposed architecture is optimized by reducing the number of the comparators. The reduction of the required comparators per process element causes that the proposed architecture loses the deterministic performance. In other words, when the partial matching is occurred, the alignment element spends additional one step in order to align the pattern. In this case, the proposed architecture could process fewer than n characters per step. It analyzes the performance of string matching architecture mathematically. When process width is n, the ratio of the matching target pattern in the whole internet packet is m and the probabilities of the each possible position of substring in the process. The experimental results show that the proposed architecture with any process width reduces the comparator requirements in comparison with the previous work.

WANG Xiaofei et.al[24] proposed a parallel Length-based matching (LBM) architecture to increase the throughput without extra memory cost. The basic idea is to process multiple characters between some specific tags in parallel. For this they use multiple hash functions solution to reduce the possibility of false positive. The evaluation shows that parallel architecture can reduce nearly 55% processing time with less memory consumption than the traditional DFA. According to proposed statistics, there are 99.41% of the input stream has been filtered by architecture which means only 0.58% (263.5M) need to be sent to StriD2FA to match instead of sending to the matching engine byte by byte. Besides the different type of traces, in this paper the other kind of trace that was collected from the World Wide Web should also be used to test the performance and stability.

HyunJin Kim and Seung-Woo Lee [25] proposed a memory-based parallel string matching engine using the compressed state transitions. In the finite-state machines of each string matcher, the pointers for representing the existence of state transitions are compressed. In addition, the bit fields for storing state transitions can be shared. Therefore, the total memory requirement can be minimized by reducing the memory size for storing state transitions. In this, four large rule sets were extracted from Snort v2.8 rules. Several parameters were swept to find their optimal values. The maximum number of states S was 128 after considering the maximum length of Snort rules. When S was 128, the number of bits in a PMV P was either 16, 24, 32, or 40; when the number of states S was 256, P was either 32, 48, 64, or 80; the maximum number of entries in the LSB transition existence table, K, and the maximum number of shared state transitions, T, were double S, respectively. In Table 1, the total memory requirements were shown by varying P. For the rule sets, the total memory requirements were minimized when S and P were 128 and 24, respectively.

Performance Evolution: The memory requirements were not minimized by only increasing P over 24; therefore, there was a threshold point of P for minimizing memory requirements. In the evaluations in which S was 256, because the required K and P increased rapidly, the required number of string matchers was not decreased. Then, by fixing S and P as 128 and 24, K was varied to find the optimal value; K was either 192, 256, 320, or 384 because K should be greater than S. Each state can have one or more than one state transition. In

these evaluations, T was fixed as 256. As a result, when K was 256, the total memory requirements were greatly reduced for all rule sets. This was mainly due to the limited number of state transitions toward noninitial states in each string matcher. With the optimal K of 256, T was varied, that is, T was either 128, 192, 256, or 320. The obtained optimal T was 256. In these evaluations, because state pointers can be shared in the transition table, the required T was small. Considering the optimal values of K and T, it is concluded that many state pointers can be shared in the transition table in the proposed string matcher.

Yi-Hua E. Yang et al.,[26] propose a novel partitioning algorithm which converts an AC-DFA into a "head" and a "body" parts. The head part behaves as a traditional ACDFA that matches the pattern prefixes up to a predefined length; the body part extends any head match to the full pattern length in parallel body-tree traversals. Taking advantage of the SIMD instructions in modern x86-64 multi-core processors, they design compact and efficient data structures packing multi-path and multi-stride pattern segments in the body-tree. Compared with an optimized AC DFA solution, their head-body matching (HBM) implementation achieves 1.2x to 3x throughput performance when the input match (attack) ratio varies from 2% to 32%, respectively. Their HBM data structure is over 20x smaller than a fully-populated AC-DFA for both Snort and ClamAV dictionaries. The aggregated throughput of their HBM approach scales almost 7x with 8 threads to over 10 Gbps in a dual-socket quad-core Opteron (Shanghai) server.

Performance Evolution: They measure the performance of their head-body matching (HBM) modules on both AMD Opteron and Intel Xeon platforms. All servers are equipped with 16 GB or more DDR2 667 MHz main memory. Due to the space limit, they only show the results on the two AMD processors. However, they note that the performance results on Clovertown is very similar to that on Barcelona, with slightly (2%–3%) greater advantage towards HBM due to Clovertown's larger cache size. They formally defined the head-body partitioned DBSM, which converts an AC-DFA into a small H-DFA and a memory-efficient tree-structured B-NFA. The H-DFA gives good cache performance on multi-core processors, while the B-NFA allows cache-friendly multi-stride transitions utilizing SIMD instructions. Unlike AC-DFA, large-scale DBSM with their head-body partitioned approach does not suffer from significant performance degradation when the match ratio increases in the input stream. The head-body approach also scales well to larger dictionaries and faster processors

3.2. PARALLEL PROCESSING BASED APPROACHES

Akhtar Rasool and Nilay Khare[27] proposed an approach, which is designed to work on SIMD parallel architecture where text is divided for parallel processing and special searching at division point is required for consistent and complete searching. This algorithm reduces the number of comparisons and parallelization improves the time efficiency. This algorithm achieves a better result as compared to the multithreaded version of the algorithm where again by text dividing, the parallelization is achieved.

cheng zhong and guo-liang chen[28] presented a perfect hash function for processing string is constructed by applying the Chinese Remainder Theorem, and a fast string matching algorithm, which is suited to process the successive sequences like the network traffic data. The determinate match results and fast execution for the string matching algorithm are very important to the network intrusion detection systems. This paper constructs a perfect hash function for processing string by applying the Chinese Remainder Theorem, transforms uniquely a pattern of length m and each substring of text of length m into a pair of integer values respectively, and presents a fast string matching algorithm which is suited to process the successive sequences like the network traffic data. The presented algorithm not only obtains the determinate match results, but also holds a linear time complexity in the worst case. The experiment results for matching a sequence database in the network intrusion detection systems also shows that the presented algorithm is efficient.

Panwei Cao and Suping Wu[29] proposed a Parallel KMP algorithm based on MPI to get higher efficiency. The tradition pattern matching algorithm need backtrack and compare repeatedly, so that affects efficiency of algorithm. Knuth and others put forward KMP algorithm in order to promote efficiency of the pattern matching. They combine MPI and KMP algorithm using MPI's Multi process to parallel KMP algorithm. By reducing the time waiting for matching, improve the string matching efficiency.

3.3. AHO CORASICK BASED APPROACHES

Many researchers propose the different hardware architectures based on the Aho-Corasick algorithm for accelerating string matching. Here we discuss some of them.

Wei Lin, Bin Liu[30] presented a pipelined parallel approach for hardware implementation of Aho-Corasick (AC) algorithm for multiple strings matching called P2-AC. P2-AC organizes the transition rules in multiple stages and processes in pipeline manner, which significantly simplifies the DFA state transition graph into a character tree that only contains forwarding edges. In each stage, parallel SRAMs are used to store and access transition rules of DFA in memory. Transition rules can be efficiently stored and accessed in one cycle. The memory cost is less than 47% of the best known AC-based methods. P2-AC supports incremental update and scales well with the increasing number of strings. By employing two-port SRAMs, the throughput of P2-AC is doubled with little control overhead.

Chuanpeng Chen and Zhongping Qin[31] proposed a high throughput configurable string matching architecture based on Aho-Corasick algorithm. The architecture can be realized by random-access memory (RAM) and basic logic elements instead of designing new dedicated chips. The bit-split technique is used to reduce the RAM size, and the byte-parallel technique is used to boost the throughput of the architecture. By the particular design and comprehensive experiments with 100MHz RAM chips, one piece of the architecture can achieve a throughput of up to 1.6Gbps by 2-byte-parallel input, and we can further boost the throughput by using multiple parallel architectures.

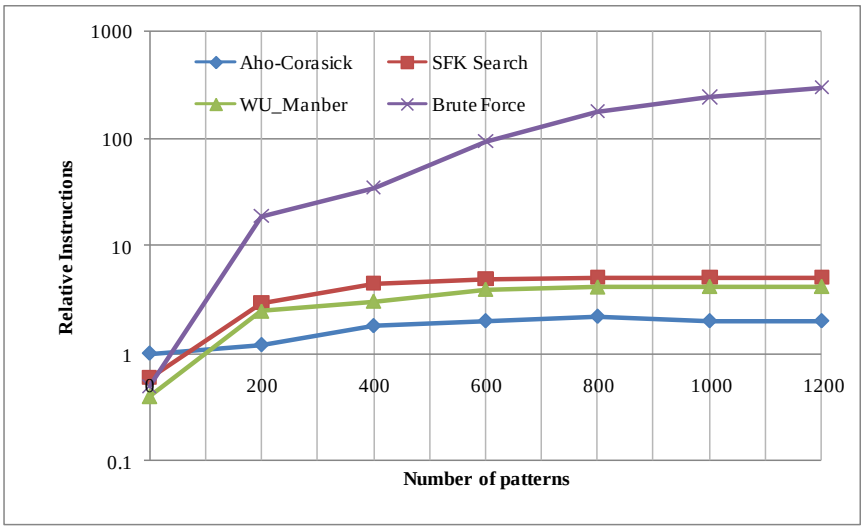

Fig 3 Number of patterns vs relative instructions

Hyun Jin Kim et.al [32] proposed an Aho-Corasick algorithm based parallel string matching. In order to balance memory usage between homogeneous finite-state machine (FSM) tiles for each string matcher, an optimal set of bit position groups is determined. Target patterns are sorted by binary-reflected gray code (BRGC), which reduces bit transitions in patterns mapped onto a string matcher. In the evaluations of Snort rules, the proposed string matching outperforms the existing bit-split string matching. Fig 3 shows the patterns vs instructions. From the Fig 4 we clearly observe that Aho-Corasick performs well when compared to other approaches[32]. Generally a Hybrid approach performs well, so from the fig 5 we observe the actual results[33].

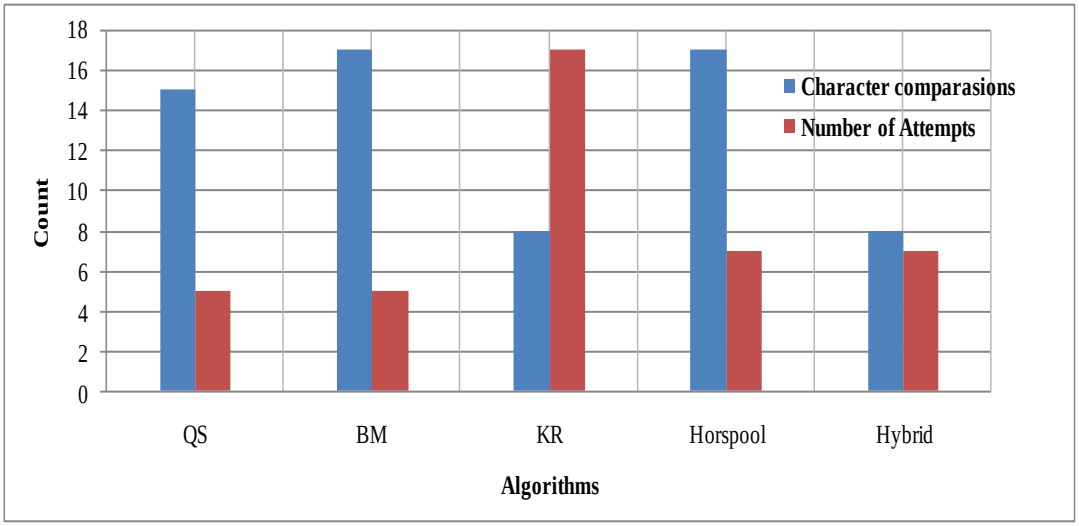

Fig 4 Comparative analysis

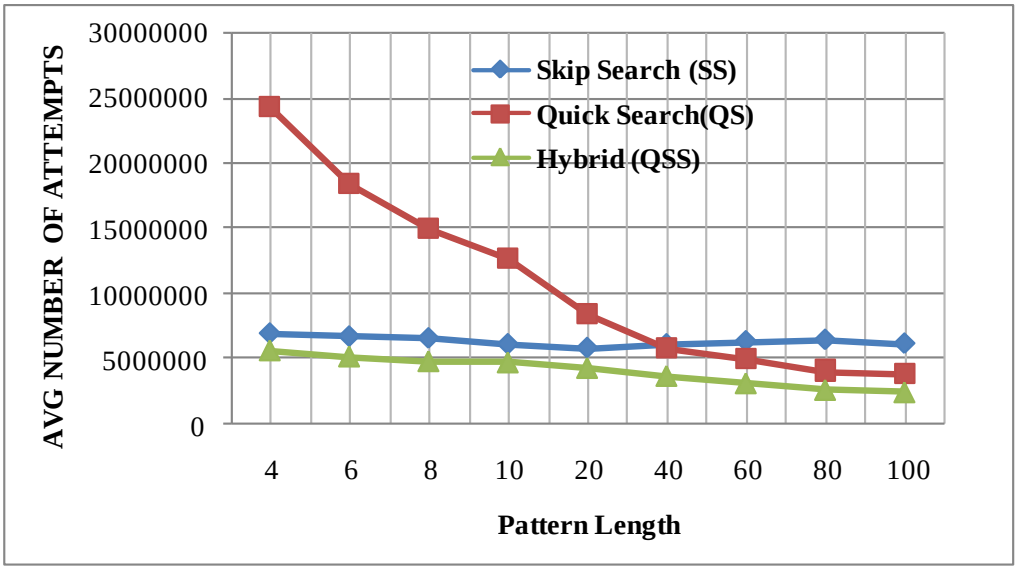

Fig 5 comparative analysis with hybrid

Alicherry et al. [34] proposed an architecture consisting of TCAM and SRAM to implement the AC algorithm that utilizes the property of ternary matching of TCAM to achieve the matching of characters expressed in negation expressions. As a result, the space required for the transitions can be reduced. Pao et al. [35] and W. Lin and B. Liu [36] proposed pipeline architectures to implement the partial trie that only contains goto functions of the AC-trie so that it can reduce the space induced by failure functions. N. Hua et al. [37] proposed another approach based on a block-oriented scheme instead of usually byte-oriented processing of patterns to reduce the memory usage. D. P. Scarpazza et al. [38] proposed an optimized software approach for a multi-core processor that splits keywords to fit in the local memories of the processing cores such that it can reach very high overall throughput. Y. Sugawara et al. [39] proposed a string matching method called suffix based traversing (SBT) that is an extension of the AC-algorithm to process multiple input characters in parallel and to reduce the size of the lookup table.

Chien-Chi Chen and Sheng-De Wang [40] proposed an architecture for multi-character transition string matching consisting of multiple matching units for processing the transition rules that are generated from the derived multi-character FSM. The design of proposed architecture utilizes the properties of the failure links of an AC-trie to reduce the transition rules derived from the failure functions linked to the initial state. As a result, the state growth rate is moderate in the number of the derived multi-character transition rules as the number of the characters inspected at a time increases. The evaluations for required space show that when the keywords are increased the complexity of AC-trie is increased rapidly, and the required space for storing the n-character rules is also increased rapidly. The results of the evaluations for required space suggest that the keywords should be partitioned to small groups and each small group of keywords is matched by small scale architecture. The proposed architecture was implemented on an ASIC device for evaluation

3.4. FINITE AUTOMATA BASED APPROACHES

HyunJin Kim et al.,[41] proposes a memory-efficient parallel string matching scheme. In order to reduce the number of state transitions, the finite state machine tiles in a string matcher adopt bit-level input symbols. Long target patterns are divided into subpatterns with a fixed length and deterministic finite automata are built with the sub patterns. Using the pattern dividing, the variety of target pattern lengths can be mitigated, so that memory usage in homogeneous string matchers can be efficient. In order to identify each original long pattern being divided, a two stage sequential matching scheme is proposed for the successive matches with sub patterns. Experimental results show that total memory requirements decrease on average by 47.8%-62.8% for Snort and ClamAV rule sets, in comparison with several existing bit-split string matching methods.

Yi-Hua E. Yang and Viktor K. Prasanna [42] proposed a head-body finite automaton (HBFA) which implements SPM in two parts: a head DFA (H-DFA) and a body NFA (B-NFA). The H-DFA matches the dictionary up to a predefined prefix length in the same way as AC-DFA, but with a much smaller memory footprint. The B-NFA extends the matching to full dictionary lengths in a compact variable-stride branch data structure, accelerated by single-instruction multiple-data (SIMD) operations. A branch grafting mechanism is proposed to opportunistically advance the state of the H-DFA with the matching progress in the BNFA. Compared with a fully-populated AC-DFA, proposed HBFA prototype has < 1/5 construction time, requires < 1/20 run-time memory, and achieves 3x to 8x throughput when matching reallife large dictionaries against inputs with high match ratios. The throughput scales up 27x to over 34 Gbps on a 32-core Intel Manycore Testing Lab machine based on the Intel Xeon X7560 processors.

Yue Hu et al.,[43] proposes a new algorithm to construct an optimal automation to achieve fast string matching. The algorithm consists of five steps: sorting, forming subtrees, encoding all subtrees, similarity checking, and completing all transitions. The algorithmic complexity is proven O(umk), where u is the number of the symbols in the alphabet set and

m and k are the average length and the number of strings being matched. They report analytical results on the matching complexity. These results prove the efficiency and effectiveness of the optimized automata generated for fast matching of multiple strings.

Junghak Kim et al.,[44] proposes a programmable string matching architecture to process multiple characters at a single cycle. To simplify the architecture of the previous works, they employ a method of realigning the input data stream by offsets. They show that some registers can be eliminated by using the method. Additionally, they present two different approaches to implement a programmable hardware for string matching on FPGA.

Yi Tang et al.,[45] proposed a paper that extends the classic longest prefix principle from single-character to multi-character string matching and proposes a multi-string matching acceleration scheme named Independent Parallel Compact Finite Automata (PC-FA). In this scheme, DFA is divided into k PC-FAs, each of which can process one character from the input stream, achieving a speedup up to k with reduced memory occupation. They introduce their observation against the prefix based automata algorithms and propose a new conception of inclusion-equivalence principle. Compared with traditional DFA approach and other improved work, PC-FA achieves a high speed-up with a lower memory cost.

Performance evolution: Experimental evaluations show that seven times of speedup can be practically achieved with a reduced memory size than up-to-date DFA-based compression approaches. They further propose a memory- efficient multi-string matching acceleration scheme named PC-FA Match Engine. Fig 6 shows the memory occupancy of the different methods[45].

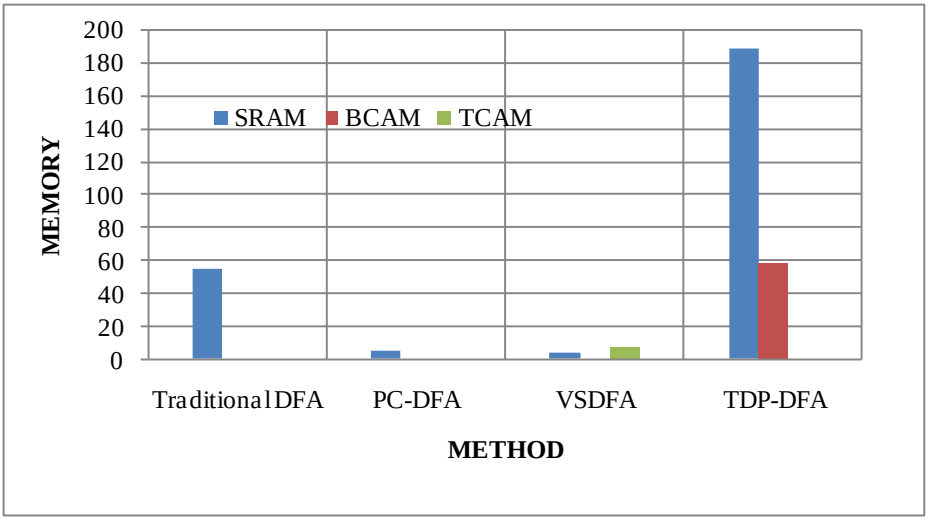

Fig 6 Memory occupancy comparisons

Xiaofei Wang et al.,[46] present stride finite automata (StriFA), a novel finite automata family, to accelerate both string matching and regular expression matching. Different from conventional finite automata, which scan the entire traffic stream to locate malicious information, a StriFA only needs to scan a partial traffic stream to find suspicious information. The presented StriFA technique has been implemented in software and evaluated based on different traces. The simulation results show that the StriFA acceleration scheme offers an increased speed over traditional nondeterministic finite automaton/deterministic finite automaton, while at the same time reducing the memory requirement.

Weirong Jiang et al., [47] generalizes the problem and proves formally that a linear pipeline with H stages can remove all cross transitions to the top H levels of a ACDFA. A novel and scalable pipeline architecture for memory efficient multi-pattern string matching is then presented. The architecture can be easily extended to support multi character input per clock cycle by mapping a compressed AC-DFA onto multiple pipelines. Simulation using Snort and ClamAV pattern sets shows that a 8-stage pipeline can remove more than 99% of the transitions in the original AC-DFA. The implementation on a state-of-the-art field programmable gate array (FPGA) shows that their architecture can store on a single FPGA device the full set of string patterns from the latest Snort rule set. Their FPGA implementation sustains 10+ Gbps throughput, while consuming a small amount of on-chip logic resources. Also desirable scalability is achieved: the increase in resource requirement of their solution is sub-linear with the throughput improvement. Simulation using real-life DPI pattern sets showed that a pipeline with no more than 8 stages could remove more than 99% of the overall transitions in a AC-DFA. They further extended the architecture to support multi-character input per clock cycle to achieve multiplicative throughput improvement. The post place and route results of the implementation on a state of- the-art FPGA showed that their design sustained over 10 Gbps throughput, while consuming a small amount of on-chip resources. Desirable scalability was exhibited: $s\times$ throughput improvement required less than $s\times$ increase in resources. The architecture is also available to support much larger pattern sets by using a limited number of external memory banks.

3.5. PREFIX BASED BASED APPROACHES

Yu Cheng and Tao Zhang [48] proposed a new string searching approach based on a data structure called prefix tree. The innovative algorithm eliminates the functional overlap of the table HASH and Prefix Function. Then they make a little improvement on the prefix tree and present a second algorithm that is faster and more space-saving. It is demonstrated analytically that the two algorithms inherit the optimality and are very competitive in practice. In order to

evaluate the performance of the proposed algorithms, experiments are carried out. They compare the algorithm against previous work, both for searching single and multiple patterns. The algorithms are tested on two kinds of texts.

Performance evaluation: Results shows the scanning time of algorithms against the size of pattern sets. In this experiment, *m* is set as 6, while |*X*| varies from 100 to 1000. The predominance of the proposed methods is notable in the English text and alphabet of size 16. In figure 7 basic and improved approaches are much faster than both the KMP and WM methods. Especially when *m* is small, the predominance of them is remarkable. With the increase of *m*, all algorithms become faster. This is because with larger *m*, they could get larger shift distances[48].

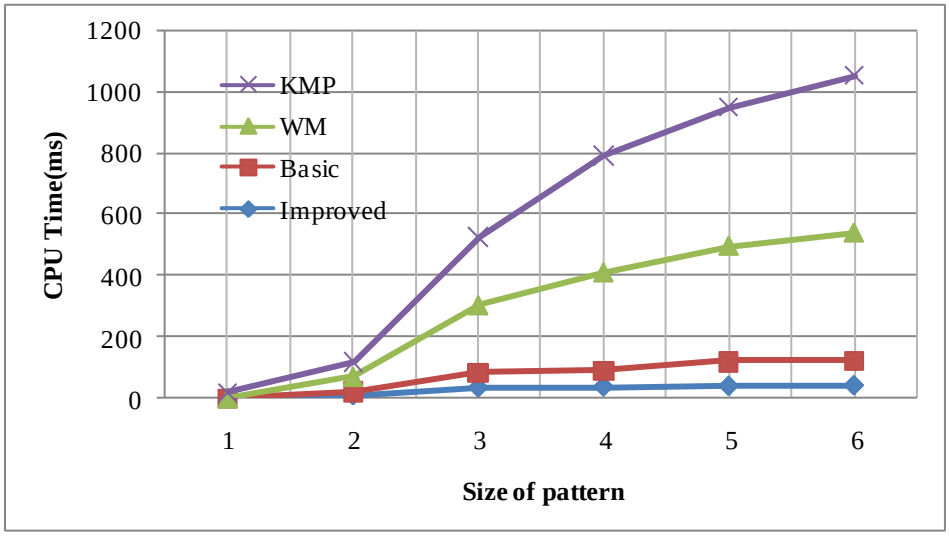

Fig 7. Test on alphabets

3.6. SUFFIX BASED APPROACHES BASED APPROACHES

Shmuel T. Klein and Dana Shapira[49] proposes an algorithm for string to dictionary matching problem based on suffix trees. The String-to-Dictionary Matching Problem is defined, in which a string is searched for all the possible concatenations of the elements of a given dictionary, with applications to compressed matching in variable to fixed length encodings. One of the motivations of their work is to enable a search directly in a text that has been compressed by some variable-to-fixed length encoding scheme, such as Tunstall's. Experiments on natural language text are presented suggesting that compressed search might use fewer comparisons for long enough patterns, in spite of a potentially large number of encodings, such as Tunstall.

3.7. HARDWARE RELATED BASED APPROACHES

KSMV Kumar et.al [50] compared string matching on single processor with multi-processors in parallel environment on hypercube network. The total time taken by search pattern is going to reduces as the No. of processors increases in network. This application developed for text documents of size only MB. It may extend to any size i.e GB to TB also and any other format likes image and video files etc. There is lot of scope to develop new trends in this area by evolving modern methods and models for increasing search speed and accuracy. To fulfill here considered both KV-KMP and KV-boyer-moore string matching algorithms for pattern matching in large text data bases using three data sets and graph's drawn for different patterns. Actual test is conducted separately for single processor, two processors, three processors and four processors. Every time, while the test is conducted the program gives elapse time for each processor separately. Therefore the average time is calculated from output result based on the maximum time taken by the individual processor among the processors involved for the particular test. The results shows that the search time taken by single processor is more when compared with multiple processors. It is also observed that as the pattern size increases the search time decreases further. For bigger pattern sizes string matching is more easier for Boyer moore algorithm because of less number of mismatches.

Yao Xin et al[51] presented a hardware architecture for the BWT-based inexact sequence mapping algorithm using the Field Programmable Gate Array(FPGA). The proposed design can handle up to two errors, including mismatches and gaps. The original recursive algorithm implementation is dealt with using hierarchical tables, and is then parallelized to a large extension through a dual-base extension method. Extensive performance evaluations for the proposed architecture have been conducted using both Virtex6 and Virtex7 FPGAs. This design is considerably faster than a direct implementation. When compared with the popular software evaluation tool BWA, their architecture can achieve the same match quality tolerating up to two errors.

Performance evaluation: Their major contributions include: (1) improving the original inefficient recursive algorithm using hierarchical tables,(2)parallelizing the inexact search process and constructing a parallel architecture by using the consecutive dual-base extension method (3)evaluating the architectures with a different number of stack arrays and processing elements(PEs). Extensive evaluation experiments are performed using both simulation datasets and real datasets. With the same inexact search options with in two errors for their architecture and BWA software, the hardware architecture can realize the same search quality as BWA. Compared with different CPU platforms running the BWA aln process, their architecture is also capable of better performance in execution speed: the Virtex6 FPGA with 2PEs implemented exceeds all software platforms except for the multithread Xeon CPU; the Virtex7 FPGA implementing 6PEs, however, can reach up to 2 times faster than the Xeon CPU with 6 threads open as shown in Fig 8 and 9[51].

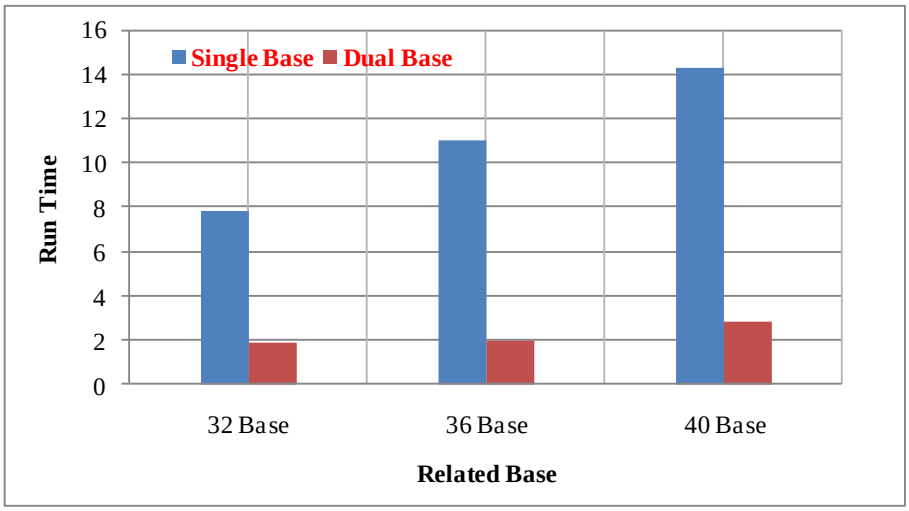

Fig 8 Single and multi base comparisons

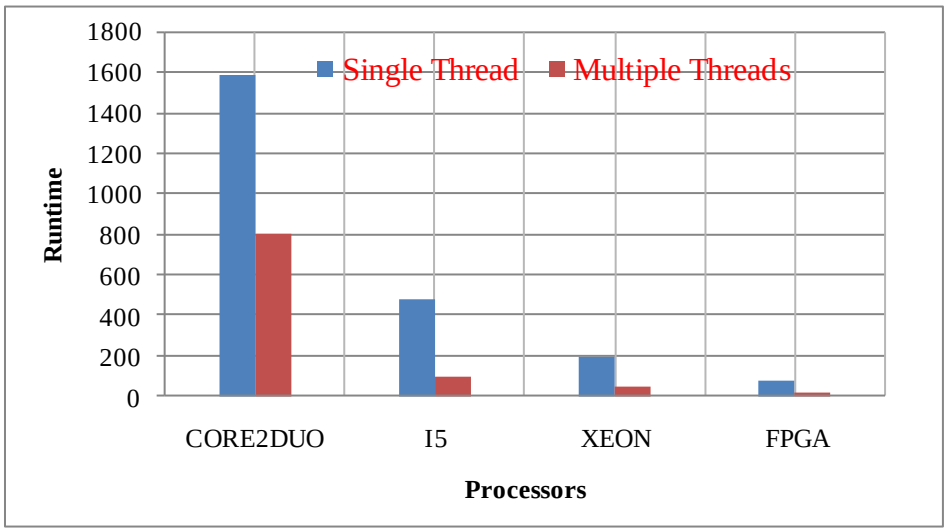

Fig 9 Different processors comparisons

Hoang Le, and Viktor K. Prasanna[52] proposed an algorithm called "leaf-attaching" to preprocess a given dictionary without increasing the number of patterns. The resulting set of post processed patterns can be searched using any tree search data structure. It also present a scalable, high-throughput, Memory-efficient Architecture for large-scale String Matching (MASM) based on a pipelined binary search tree. The proposed algorithm and architecture achieve a memory efficiency of 0.56 (for the Rogets dictionary) and 1.32 (for the Snort dictionary). As a result, our design scales well to support larger dictionaries. Implementations on 45 nm ASIC and a state-of-the-art FPGA device (for latest Rogets and Snort dictionaries) show that proposed architecture achieves 24 and 3.2 Gbps, respectively. The MASM module can simply be duplicated to accept multiple characters per cycle, leading to scalable throughput with respect to the number of characters processed in each cycle. Dictionary update involves simply rewriting the content of the memory, which can be done quickly without reconfiguring the chip.

Jiaying Wang et al.,[53] study string similarity search and join with edit distance constraints. Although multicore machines have become the mainstream computer architecture, most existing methods only work on a uniprocessor. To address this problem, they propose a novel parallel framework using BWT. They also devise efficient technique to utilize cache to further speed up the performance. Their method can solve similar search and join efficiently and generally. They conducted a comprehensive experimental study of the method to demonstrate the efficiency.

Performance evolution: They proposed a cache-aware parallel method using BWT to find similar strings in text collection. They devised a new index called BWTPA. They used BWTPA to index strings, and it partitioned queries to segments. They developed an algorithm to search these segments to find candidates efficiently and verified the candidates to get the results. They also developed several optimization strategies to improve the performance. They proposed two methods to do similarity join on the index. They have implemented their algorithms, and experiments showed the efficiency of their method.

Christoph Strecha et al.,[54] map the descriptor vectors into the Hamming space, in which the Hamming metric is used to compare the resulting representations. This way, they reduce the size of the descriptors by representing them as short binary strings and learn descriptor invariance from examples. Feature descriptors can be designed to be invariant to certain classes of photometric and geometric transformations, in particular,

affine and intensity scale transformations. The real transformations that an image can undergo can only be approximately modeled in this way, and thus most descriptors are only approximately invariant in practice, the descriptors are usually high-dimensional.

Edward Fernandez et al.,[55] proposed the first FPGA-based hardware implementation of the FM-index for exact pattern matching.

Performance evolution: They evaluate the performance of the FM-index, first by comparing it to a brute force implementation and second by evaluating the expected number of character matches that would performed for each search pattern. Fig 9 shows execution times of Bowtie and FGPA implementation with the number of matching DNA sequences on the E-coli genome in percent. In this study they first describe the implementation of the FM-index in hardware, it is compared to the brute force approach and it is shown that the FM-index has a higher effective throughput than the brute force. This is due to the higher number of character comparisons per cycle performed by the FM-index even though it operates on a lower operating. They report experimental results on the problem of mapping short DNA sequences to a reference genome. They show that the throughput of the FM-index is significantly higher than the naïve (brute force) approach. Like the Bowtie software tool, the FM-index can abandon early the hardware matching. It outperforms Bowtie by two orders of magnitude which is shown in Fig 10[55].

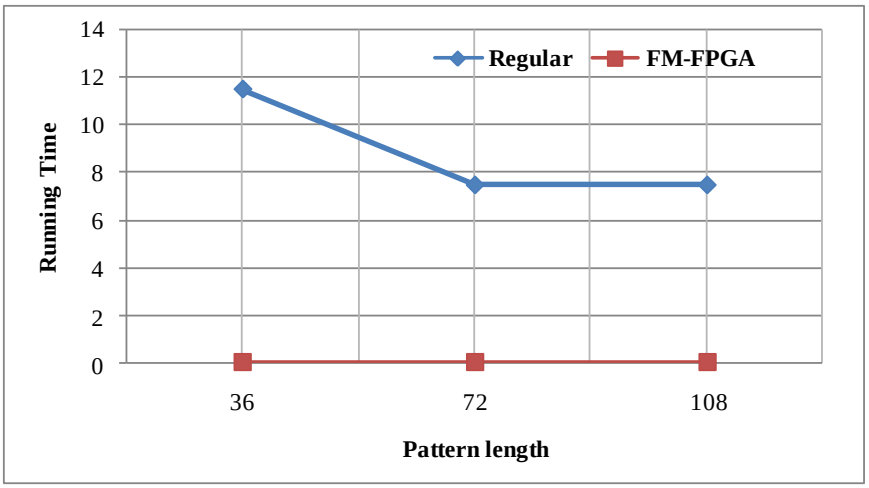

Fig 10 Pattern length vs running time

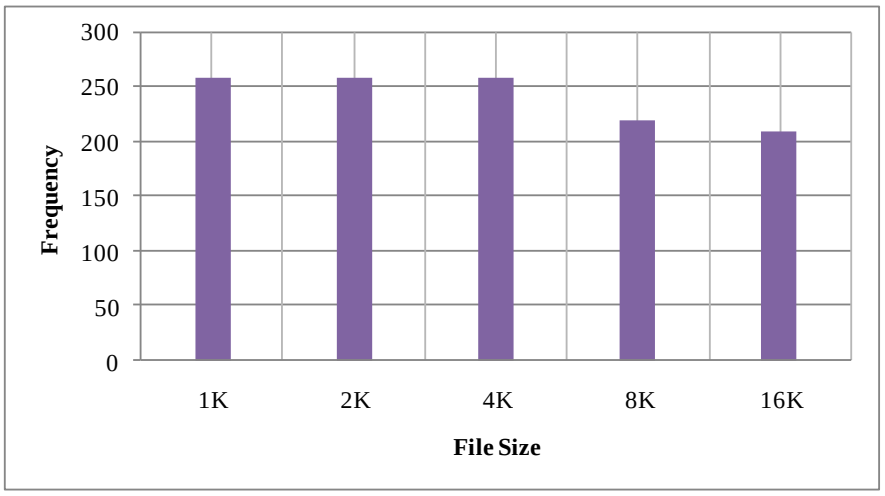

Fig 11 file size vs frequency

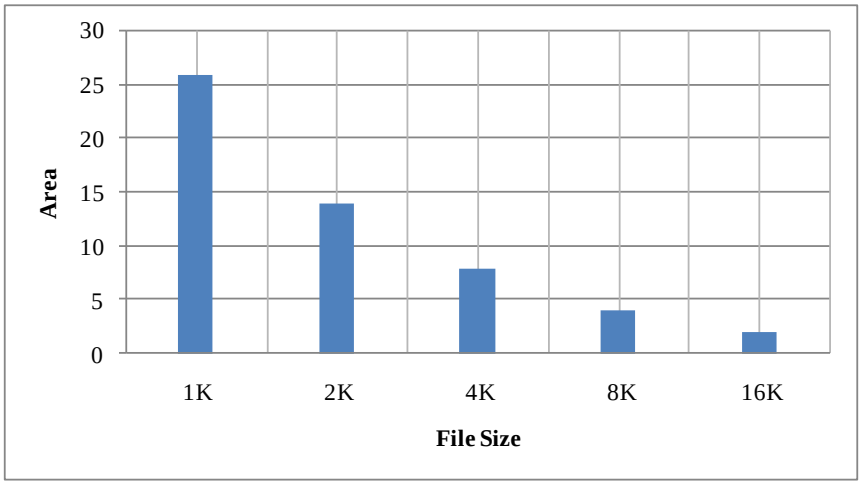

Figure 12 file size and area

They synthesized with place and route implementations on a Xilinx Virtex 6 (XC6VLX760) FPGA. Figure 10, 11 and 12 shows the operating frequency and slices utilized (as % of total slices) for increasing text section length per module. Increasing the section size, from 1K to 16K characters, decrease the operating frequency by 20%. This is due to larger adder sizes and larger module circuits. The percentage of slices required for these decreases from 25% to 2.5%. This is because more logic is required for processing multiple FM-indices compared to a single unit for one long text section length. Looking at both graphs, it is better to implement the FM-index on an FPGA using longer text lengths[55].

TAN Jianlong et al.,[56] proposes a novel method to extract the partial strings from each pattern which maximizes search speed. More specifically, with this method they can compute all the corresponding searching time cost by theoretical derivation, and choose the location which yields an approximately minimal search time.. String matching plays a key role in web content monitoring systems. Suffix matching algorithms have good time efficiency, and thus are widely used. These algorithms require that all patterns in a set have the same length. When the patterns cannot satisfy this requirement, the leftmost m-characters, m being the length of the shortest pattern, are extracted to construct the data structure. They call such m-character strings as partial strings. However, a simple extraction from the left does not address the impact of partial string locations on search speed. They evaluate their method on two rule sets: Snort and ClamAV. Experiments show that in most cases, their method achieves the fastest searching speed in all possible locations of partial string extraction, and is about 5%-20% faster than the alternative methods.

Prasad et.al.,[57] propose two new bit-parallel algorithms to solve the same problem. These new algorithms requires no verification and can handle patterns of length > w. These two techniques also use the same BPA of approximate matching and concatenation to form a single pattern from the set of r patterns. It compares the performance of new algorithms with existing algorithms and found that proposed algorithms MASM1 and MASM2 have better running time than the existing algorithms: MASM and BPA (running r times). According to Fig 13 The running time of both MASM1 and MASM2 increases: (i) on increasing the maximum error allowed and, (ii) pattern length. The running time of MASM1 Algorithm is better than that of MASM2 Algorithm. Hence it can be concluded that proposed is better among the other approaches[57].

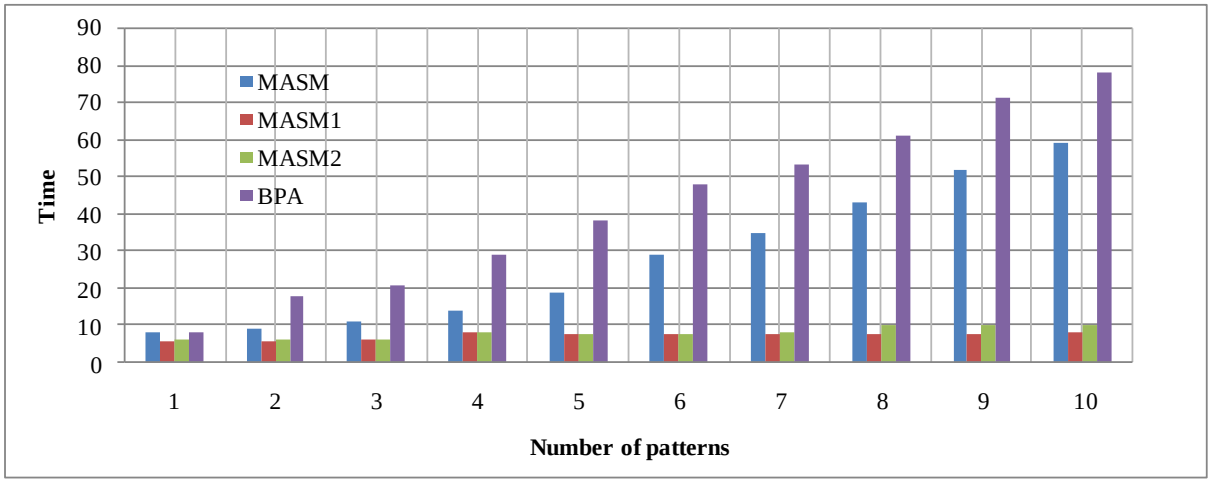

Fig 13 Number of patterns vs Time for different methods

Mosleh M. Abu-Alhaj et al.,[58] proposes a general platform for improving the existing Exact String-Matching algorithms executing time, called the PXSMAlg platform. The function of this platform is to parallelize the Exact String-Matching algorithms using the MPI model over the Master/Slaves paradigms. The PXSMAlg platform parallelization process is done by dividing the Text into several parts and working on these parts simultaneously. This improves the executing time of the Exact-String- Matching algorithms. They have simulated the PXSMAlg platform in order to show its competence, through applying the Quick Search algorithm on the PXSMAlg platform. The simulation result showed significant improvement in the Quick Search executing time, and therefore extreme competence in the PXSMAlg platform.

PXSMAlg Platform Performance Analysis: They have built a simulation to demonstrate the feasibility of the PXSMAlg platform and its compatibility with the Exact-String-Matching algorithms. The simulation is done to compare the performance of the PXSMAlg platform with the conventional method, that is, the sequential method. The simulation built is based on three main factors: executing time, speedup, and efficiency. Their simulation runs under the Aurora server, which consists of 14 nodes, with each node having 2 CPUs, a speed of 1300MHz and a 1GB memory; all nodes run the Linux OS. The results showed high performance of the PXSMAlg platform over the sequential methods. They have carried out 14 different experiments to search for the letter "a" in a 37 MB file size which is shown in Fig 14 and 15[59]

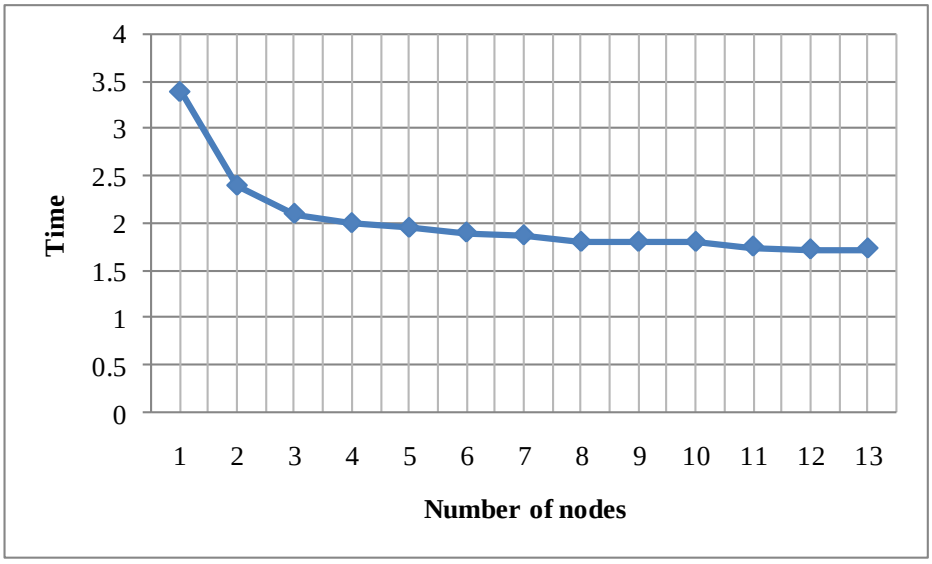

Fig 14 Execution time Comparison

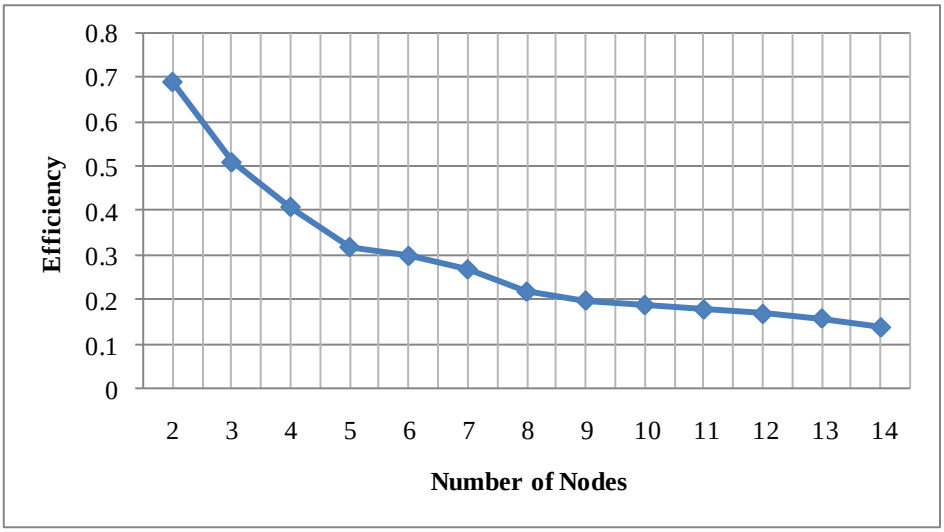

Fig 15 Efficiency Comparison

3.8. GPU'S BASED APPROACHES

Benedikt Forchhammeret.al [60] presented a complete duplicate detection workflow that utilizes the capabilities of modern graphics processing units (GPUs) to increase the efficiency of finding duplicates in very large datasets. Proposed solution covers several well-known algorithms for pair selection, attribute-wise similarity comparison, record-wise similarity aggregation, and clustering. Here redesigned these algorithms to run memory-efficiently and in parallel on the GPU. Proposed experiments demonstrate that the GPU-based workflow is able to outperform a CPU-based implementation on large, real-world datasets. For instance, the

GPU-based algorithm deduplicates a dataset with 1.8m entities 10 times faster than a common CPU-based algorithm using comparably priced hardware.

Antonino Tumeo et al.,[61] focus on the matching of unknown inputs streamed from a single source, typical of security applications and difficult to manage since the input cannot be preprocessed to obtain locality. They consider shared-memory architectures (Niagara 2, x86 multiprocessors and Cray XMT) and distributed memory architectures with homogeneous (InfiniBand cluster of x86 multicores) or heterogeneous processing elements (InfiniBand cluster of x86 multicores with NVIDIA Tesla C1060 GPUs).

Performance Evolution: They have presented several software implementations of the Aho-Corasick pattern matching algorithm for high performance systems, and carefully analyzed their performance. They presented optimized designs for the various architectures, discussing several algorithmic strategies, for shared memory solutions, GPU-accelerated systems and distributed memory systems. They describe how each solution achieves the objectives of supporting large dictionaries, sustaining high performance, and enabling customization and flexibility using various data sets. They found that the absolute performance obtained on the Cray XMT is one of the highest reported in literature, at ≈ 28 Gbps (using 128 processors) for a software solution with very large dictionaries. This work compares several software-based implementations of the Aho-Corasick algorithm for high performance systems.

Antonino Tumeo et.al [62] presented several software implementations of the Aho-Corasick pattern matching algorithm for high performance systems, and carefully analyzed their performance. It considered the various tradeoffs in terms of peak performance, performance variability, and data set size. It presented optimized designs for the various architectures, discussing several algorithmic strategies, for shared-memory solutions, GPU-accelerated systems, and distributed-memory systems. Finally from this paper found that the absolute performance obtained on the Cray XMT is one of the highest reported in the literature, at 28 Gbps (using 128 processors) for a software solution with very large dictionaries. Through multithreading and memory hashing the XMT is able to maintain stable performance across very different sets of dictionaries and input streams. A dual Niagara 2 obtains stable performance only in low and medium matching conditions, while a dual Xeon 5560 has more varied results, obtaining high peak rates for light matching conditions, but progressively reducing its performance as the number of matches' increases.

3.9. RAM BASED APPROACHES

Vinod.O et.al [63] proposes an alternative algorithm using a Hash Function which uses a SRAM that creates fingerprints of the packet payload which are then compared with the patterns signatures. The proposed hash based system consumes around 0.56 times or 56 percent less memory than the memory consumed by the RTCAM method. It can also be observed from the results that as the TCAM width doubles the initial width the memory consumption increases around 1000kb the initial memory consumption value in RTCAM method. But in the case of hash based method as the block size is doubled the memory consumption increases by a small value around 200kb only from the initial memory consumption value. Hence the proposed hash based method is efficient than the RTCAM method in terms of memory consumption.

Oren Ben-Kiki et al.,[64] proposed macro-level algorithm only uses the standard AC instructions of the word-RAM model (i.e. no integer multiplication) plus two specialized micro-level AC ^{0} word-size packed-string instructions. The main word-size string matching instruction wssm is available in contemporary commodity processors. The other word-size maximum-suffix instruction wslm is only required during the pattern pre-processing.

Performance avolution: They demonstrated how to employ word-size string matching instructions to design optimal packed string matching algorithms in the word-RAM, which are fast both in theory and in practice They also consider the complexity of the packed string matching problem in the classical word-RAM model in the absence of the specialized micro-level instructions wssm and wslm.They propose micro-level algorithms for the theoretically efficient emulation using parallel algorithms techniques to emulate wssm and using the Four-Russians technique to emulate wslm. Surprisingly, bit-parallel emulation of wssm also leads to a new simplified parallel random access machine string matching algorithm. As a byproduct to facilitate their results they develop a new algorithm for finding the leftmost (most significant) 1 bit in consecutive non-overlapping blocks of uniform size inside a word.

3.10. APPROACHES FOR GENOME SEQUENCES

More approaches have been developed for the k differences problem in fields including molecular biology. Some of them are briefly mentioned in this part.

Cheng and Fu [65] proposed VLSI architecture of two dimensional arrangements of n*m processing elements. P-NAC (Princeton Nucleic Acid Comparator) [66] was built using linear systolic array architecture for comparing DNA sequences.

Similarly, Sastry et al [67] presented a VLSI chip for computing similarity between two strings. Two generations of the Splash processors, which are based on systolic arrays of FPGAs (field-programmable gate arrays) have been designed [68]. As Foster and Kung [69] 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. This matter is also applicable to the dataflow environment. As used in most high performance software string matching algorithms, trial of skipping operations will degrade the overall performance of the dataflow algorithm. Thus we start from the naïve algorithm which requires n-m+1 attempts and each attempt takes m comparisons.

Carla Correa Tavares dos Reis and Oswaldo Cruz[70] presented the development of algorithms for approximate string matching using parallel methods. It intends to do the maximum of molecular sequences comparisons per unity of time. The parallel program implementation has carried out in C on an available twenty processing nodes clustering architecture using a model of parallel programming systems, the MPI (Message-Passing Interface), which is as library of subroutines. In this paper also concerned with reporting the speedup and efficiency measures. More precisely, present a parallel algorithm for approximate sequence matching, showing its implementation and reporting its measures in

comparison to its sequential version. It use one of the possible approaches to reduce the time spent on comparisons of molecular database sequences by distributing the data among processors, which achieves a linear speedup (time) and requires constant space memory per processor. It also compares between the serial processing and the parallel processing (under the operation conditions offered by MPI ambient), the parallel version always gave the best results (execution and data distribution times).

Muhammad Zubair et al.,[71] propose a new concept to solve the problem of exact string matching by scanning text string for the rightmost character of the pattern in preprocessing phase. In matching phase TSPRC (Test Scanning for Pattern Rightmost Character) compares the pattern with text window from both directions simultaneously. They proposed a new algorithm TSPRC,in addition to proposed algorithm Naive, Not So Naive, quick Search, Boyer Moor Bad Character and Berry Ravindran algorithms are experimented with TSPRC. In the experiment they took a text string T of the size of sixty thousand characters and pattern P of lengths {6, 12, 18, 24, 30, 36, 42, 48, 54, 60}. Text String is consisted of the four characters L= {A, C, G, T} these are the characters occurred in DNA pattern. Pattern is also of L={A, C, G, and T}. Several experiments have been conducted and the obtained results are compared with Naive, Not So Naive, Quick Search and Berry Ravndrn algorithms. Comparison made on two bases; total number of characters compared by each algorithm and the number attempts taken by each algorithm for finding all possible occurrence of the pattern in the text.

Performance evolution: Comparison of proposed algorithm is made with existing algorithms on the bases of the number of characters compared and the attempts made by experimented algorithms to complete the task. In preprocessing phase of TSPRC; rightmost character of pattern is searched in the text string. Index of the character in the text string is used to calculate the shift's length and align the pattern with next text window. The analysis and the experimental results illustrate that the TSPRC algorithm is better than the number of existing algorithms, which is shown in Fig 16 [71].

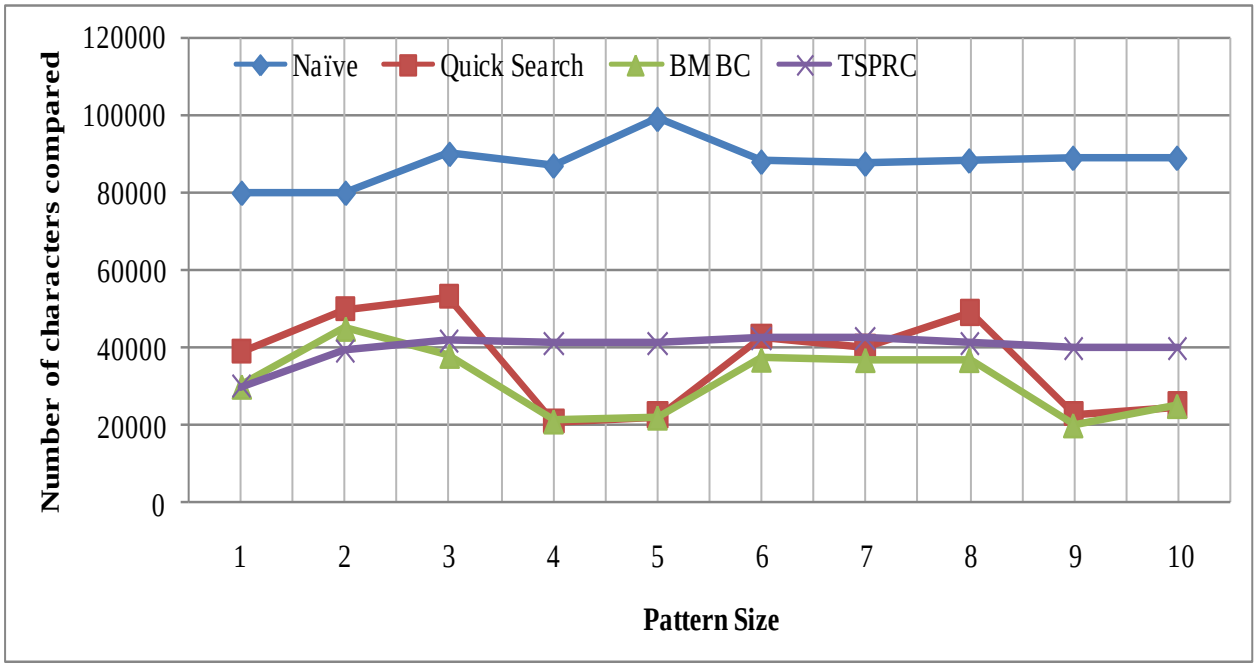

Fig 16 Pattern size vs number of characters compared

Yunho Oh et al.,[72] proposed a new parallel genome matching algorithm using graphics processing units (GPUs). They proposed approach is based on the Aho–Corasick algorithm and it was developed based on a consideration of the architectural features of existing GPUs with a hundred or more cores. They provide an appropriate task partitioning method that runs on multiple threads and they fully utilize the cache memory and the shared memory structures available in GPUs and also propose a tiled access method for rapid data transfer from the global memory to the shared memory. They also provide new models for cache-friendly state transition table to improve performance of pattern matching operations on GPUs.

Performance evolution: They have proposed a parallelized genome pattern matching algorithm that considered the memory structure of GPUs. In the GPGM algorithm, the tiled access using a 2D thread array was proposed to eliminate data transpose operations. In addition, the proposed algorithms successfully improved the performance while reducing the STT structure. For that purpose, they introduced the ML structure with a vector data type, which facilitated 13.3% faster pattern matching performance due to its highly efficient cache utilization. The maximum Throughput they achieved in various experiments was 15.3Gbps. Moreover, they showed that their proposed design outperformed an earlier approach with a 15.4%performance improvement which will shown in Fig 17[72].

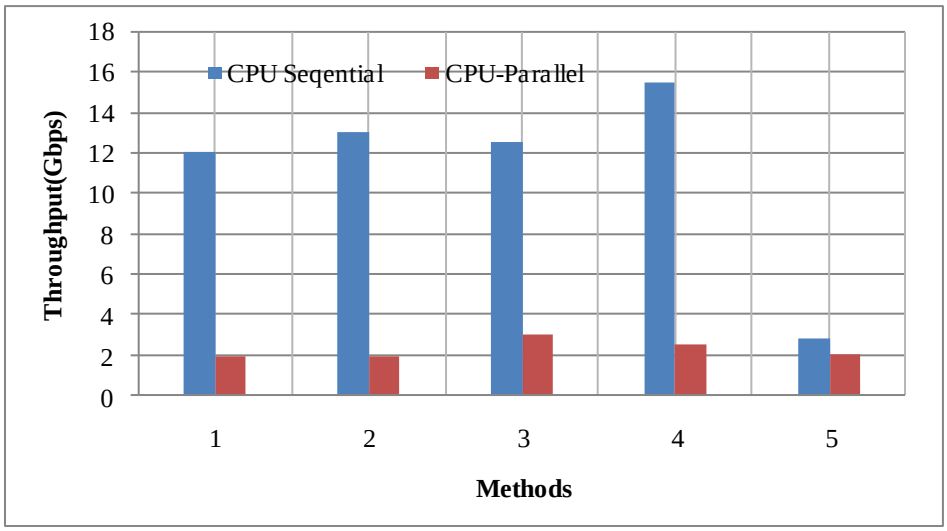

Fig 17 Performance evolution different methods

Tomohiro I et al.,[73] presented a linear-time algorithm to solve the palindrome pattern matching problem. The first algorithm is a Morris–Pratt type algorithm, and the second one is a suffix-tree type algorithm. The palindrome pattern matching problem is to compute all positions i of t such that Pals(p) = Pals(t[i : i + m − 1]), given a text t of length n and a pattern p of length m. Palindromes in strings have widely been studied both in theoretical and practical contexts, such as in word combinatorics and in bioinformatics In practical applications such as DNA and RNA sequence analysis, it is desired to cope with gapped palindromes which have a spacer between the left and right arms of the palindromes .

3.11. OTHER APPROACHES

Fang Xiangyan et al.,[74] proposed improving methods to advance the matching rate for multi-pattern string matching algorithm Wu-Manber, First, string abstract value matching method advances the precision of the first matching, and reduces the string comparing times, second, heuristic matching method increases the safe moving distance when the matching of string matched is successful, third, multi-level cache parallel matching method removes the delay time of the long pattern string matching and hash operation. The main advantages for W-M algorithm are big skip distance and lessen string comparing number. Therefore, the efficiency of W-M algorithm is higher than Aho Corasick algorithm in actual application. Lastly, the algorithmic complexity is analyzed.

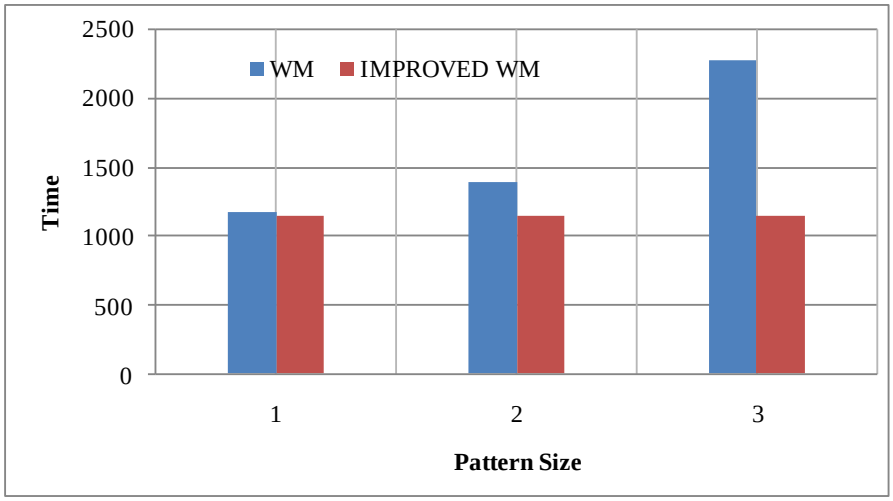

Fig 18 pattern size vs Execution time

Performance evolution The string matched will be compared with the pattern string reading from the memory when the suffix sub-string of string matched is exactly matched with pattern string in W -M algorithm, which judges that they are exactly matched. The abstract value matching method will filter many strings which is not exactly matched, reduce the accessing to slow memory and string comparison operations, especially reduce the long pattern string comparison, and advance the accuracy of the first matching in improving algorithm. The abstract value calculation method has MD5, SHA-l , HASH and so on, which require more computing time. XOR is simple operation and is good method of calculation abstract. String matched with length m do XOR to produce a 32-bit abstract value according to the order. FLAG

table is used to store the result of abstract value matching which will show in Fig 18[74].

Zhengda Xiong [75] gives analysis to several classical algorithms, KMP, BM and their improvements. Then, by compositing the main method of the BM algorithm. Then, by compositing the main method of the BM algorithm they propose a new algorithm called Composite BM algorithm (CBM). Differing from the BM algorithm that only uses current matching information, the CBM algorithm tries to take full advantage of the historical matching information. In this manner, CBM accelerates the forward speed of the pattern during the matching, and hence the whole string matching process is speeded up effectively.

Performance evolution: For binary matching, they made random test to BM and CBM, the result shown the efficiency of CBM is higher than of BM which is shown in Fig 19[75].

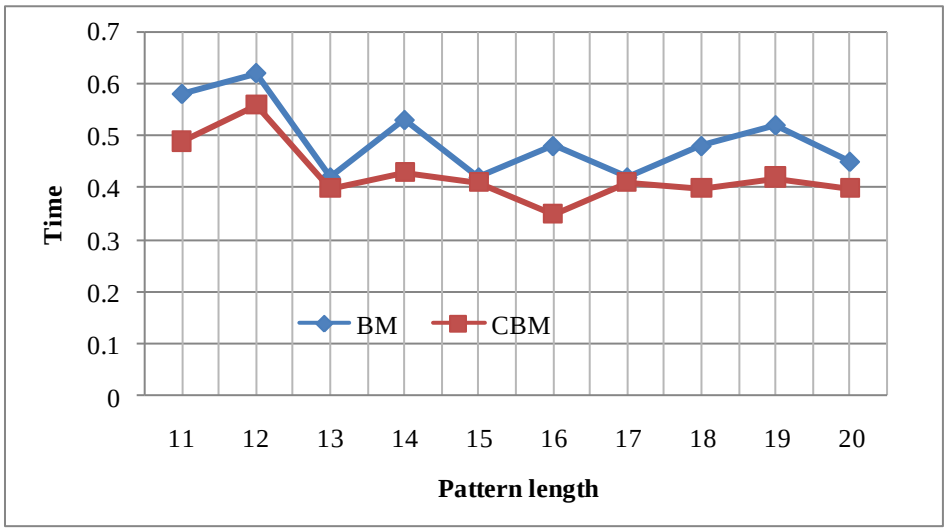

Fig 19 comparasion of BM and CBM

GolnazBadkobeh et.al [76] Proposed an index constructed directly from the run-length encoding of *s*. The construction time of our index is $O(n+\rho 2\log\rho)$,where $O(n)$ is the time for computing the run-length encoding of *s* and ρ is the length of this encoding this is no worse than previous solutions if $\rho=O(n/\log n)$ and better if $\rho=o(n/\log n)$.Our index *L* can be queried in $O(\log\rho)$ time. While $|L|=O(\min(n,\rho 2))$ in the worst case, preliminary investigations have indicated that |*L*| may often be close to ρ.Further more,the algorithm for constructing the index is conceptually simple and easy to implement.

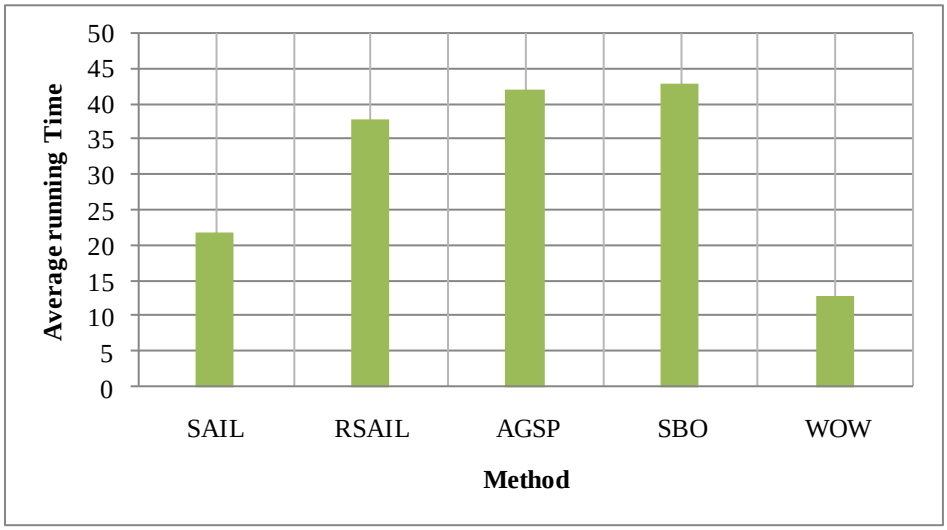

Fig 20 Method vs avg running time

Dan Guo et al.,[77] focuses on the problem with gap-length constraints and the one-off condition (The one-off condition means that each character can be used at most once in all occurrences of a pattern in the sequence). It is difficult to achieve the optimal solution. They propose a graph structure WON-Net to obtain all candidate matching solutions and then design the WOW algorithm with the weighted centralization measure based on nodes' centrality-degrees. They also propose an adjustment mechanism to balance the optimal solutions and the running time. They also define a new variant of WOW as WOW-δ. They have formalized the PMGO problem with additions of flexible sub-patterns, wildcards and the one-off condition to pattern matching, and these additions also bring difficulty on achieving the optimal solutions. They proposed the graph structure WON-Net and the WOW algorithm. They have provided six conditional completeness theorems of matching solutions based on the graph structure WON-Net and the WOW algorithm. They also proposed a

variant of WOW (the WOW-δ algorithm) to keep a balance between running time and solutions

Performance evolution: Theoretical analysis and experiments demonstrate that WOW and WOW-δ are more effective than their peers Fig 20[77].

4. CONCLUSIONS

In this study, we widely investigate the problem of parallel string matching in the context of Parallel Computing. An overview of string matching is made, in which the different forms of parallel string matching problem are also distinguished, and the classifications of parallel string matching problem are discussed. We critically reviewed different classifications of parallel string matching algorithms. Based on this study, a number of constructive suggestions are made which will helpful to the researchers for developing better techniques.

5. REFERENCES

[1] Chinta Someswararao, K Butchiraju, S ViswanadhaRaju, "Recent Advancement is Parallel Algorithms for String matching on computing models - A survey and experimental results", LNCS, Springer,pp.270-278, ISBN: 978-3-642-29279-8 ,2011.

[2] Chinta Someswararao, K Butchiraju, S ViswanadhaRaju, "PDM data classification from STEP- an object oriented String matching approach", IEEE conference on Application of Information and Communication Technologies, pp.1-9, ISBN: 978-1-61284-831-0, 2011.

[3] Chinta Someswararao, K Butchiraju, S ViswanadhaRaju, "Recent Advancement is Parallel Algorithms for String matching - A survey and experimental results", IJAC, Vol 4 issue 4, pp-91-97, 2012.

[4] Simon Y. and Inayatullah M., "Improving Approximate Matching Capabilities for Meta Map Transfer Applications," Proceedings of Symposium on Principles and Practice of Programming in Java, pp.143-147, 2004.

[5] Chinta Someswararao, K Butchiraju, S ViswanadhaRaju, "Parallel Algorithms for String Matching Problem based on Butterfly Model", pp.41-56, IJCST, Vol. 3, Issue 3, July – Sept, ISSN 2229-4333, 2012.

[6] Chinta Someswararao, K Butchiraju, S ViswanadhaRaju, "Recent Advancement is String matching algorithms- A survey and experimental results", IJCIS,Vol 6 No 3, pp.56-61, 2013.

[7] S. viswanadha raju,"parallel string matching algorithm using grid", "international journal of distributed and parallel systems" (ijdps) vol.3, no.3, 2012**.**

[8] Leslie G. Valiant, A bridging model for parallel computation, Commun. ACM, volume 33, issue 8, August, 1990, pages 103—111

[9] I. Foster. Designing and Building Paral lel Programs. Addison Wesley, 1996.

[10] I. Foster and S. Tuecke. Parallel Programming with PCN. Technical Report ANL-91/32, Argonne National Laboratory, Argonne, December 1991.

[11] J. Darlinton, M. Ghanem, H. W. To (1993), "Structured Parallel Programming", *In Programming Models for Massively Parallel Computers. IEEE Computer Society Press. 1993*

[12] 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.

[13] 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.

[14] 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.

[15] 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.

[16] J.Nandhini, Dr.M. Nithya, Dr.S.Prabhakaran "Advance virus detection using combined techniques of pattern matching and dynamic instruction sequences", International Journal of Communication and Computer Technologies, Volume 01 – No.45, pp.156-161 2013

[17] Safaa O. Al-Mamory et al.," String Matching Enhancement for Snort IDS" , pp.1020-1023.

[18] S. Dharmapurikar and J.W. Lockwood, "Fast and scalable pattern matching for network intrusion detection system", *IEEE JSAC*, Vol.24, No.10, pp.1781–1792, 2006.

[19] N. Hua, H. Song and T.V. Lakshman, "Variable-stride multipattern matching for scalable deep packet inspection", in *Proc. of INFOCOM*, Rio de Janeiro, Brazil, pp.415–423, 2009.

[20] H. Lu, K. Zheng, B. Liu, X. Zhang and Y. Liu, "A memoryefficient parallel string matching architecture for high-speed intrusion detection", *IEEE JSAC*, Vol.24, No.10, pp.1793–1804, 2006.

[21] B.C. Brodie, D.E. Taylor and R.K. Cytron, "A scalable architecture for high-througshput regular-expression pattern matching", in *Proc. Of ACM/IEEE International Symposium on Computer Architecture* (*ISCA*), Boston, MA, USA, pp.191–202, 2006.

[22] D. Ficara, G. Antichi, A.D. Pietro, S. Giordano, G. Procissi and F. Vitucci, "Sampling techniques to accelerate pattern matching in network intrusion detection systems", in *Proc. of IEEE ICC*, Cape Town, South Africa, pp.1–5, 2010.

[23] Seongyong Ahn, Hyejong Hong, Hyunjin Kim, Jin-Ho Ahn, Dongmyong Baek and Sungho Kang, "A Hardware-Efficent Multi-character String Matching Architecture Using Brute-force Algorithm, ISOCC, pp. 464-467, 2009.

[24] WANG Xiaofei, HU Chengchen, TANG Yi, ZHANG Ting, WU Chunming, LIU Bin and WANG Xiaojun, "Parallel Length-based Matching Architecture for High

Throughput Multi-Pattern Matching", Chinese Journal of Electronics, Vol.21, No.3, pp.489-494,2012.

[25] HyunJin Kim and Seung-Woo Lee, "A Hardware-Based String Matching Using State Transition Compression for Deep Packet Inspection", ETRI Journal, Volume 35, Number 1,pp.154-157 ,2013

[26] Yi-Hua E. Yang et al.," Head-Body Partitioned String Matching for Deep Packet Inspection with Scalable and Attack-Resilient Performance" , IEEE, 2010.

[27] Akhtar Rasool and Nilay Khare, "Parallelization of KMP String Matching Algorithm on Different SIMD architectures-Multi-Core and GPGPU", International Journal of Computer Applications, pp.26-28, 2012.

[28] cheng zhong and guo-liang chen, " a fast determinate string matching algorithm for the network intrusion detection systems", Proceedings of the Sixth International Conference on Machine Learning and Cybernetics, Hong Kong, pp.,3173-3177, 2007

[29] Panwei Cao and Suping Wu , "Parallel Research on KMP Algorithm", pp.4252-4255, IEEE, 2011.

[30] Wei Lin, Bin Liu , "Pipelined Parallel AC-based Approach for Multi-String Matching", 14th IEEE International Conference on Parallel and Distributed Systems, pp. 665- 672,2008.

[31] Chuanpeng Chen and Zhongping Qin A Bit-split Byte-parallel String Matching Architecture, IEEE ,pp.214-217, 2009.

[32] HyunJin Kim, Hyejeong Hong, Hong-Sik Kim, and Sungho Kang, "A Memory-Efficient Parallel String Matching for Intrusion Detection Systems", IEEE COMMUNICATIONS LETTERS, VOL. 13, NO. 12, pp. 1004-1006, 2009.

[33] Mustafa Abdul Sahib Naser, Nur'Aini Abdul Rashid, and Mohammed Faiz Aboalmaaly, "Quick-Skip Search Hybrid Algorithm for the Exact String Matching Problem" , International Journal of Computer Theory and Engineering Vol. 4, No. 2, pp.259-265, 2012

[34] M. Alicherry, M. Muthuprasanna and V. Kumar, High speed pattern matching for network IDS/IPS, *IEEE ICNP*, pp.187-196, 2006.

[35] D. Pao, W. Lin and B. Liu, A memory-efficient pipelined implementation of the Aho-Corasick string- matching algorithm, *ACM Trans. on Archit. Code Optim.*, vol.7, pp.1-27, 2010.

[36] W. Lin and B. Liu, Pipelined parallel AC-based approach for multi-string matching, *IEEE ICPADS*, pp.665-672, 2008.

[37] N. Hua, H. Song and T. V. Lakshman, Variable-stride multi-pattern matching for scalable deep packet inspection, *IEEE INFOCOM*, pp.415-423, 2009.

[38] D. P. Scarpazza, O. Villa and F. Petrini, Exact multi-pattern string matching on the cell/b.e. processor, *ACM CF*, 2008.

[39] Y. Sugawara, M. Inaba and K. Hiraki, Over 10Gbps string matching mechanism for multi-stream packet scanning systems, *Field Programmable Logic and Application*, vol.3203, pp.484-493, 2004.

[40] Chien-Chi Chen and Sheng-De Wang , "A Multi-Character Transition String Matching Architecture Based On Aho-Corasick Algorithm", *International Journal of Innovative Computing, Information and Control, pp.* 8367-8386,2012.

[41] HyunJin Kim et al.," A Memory-Efficient Bit-Split Parallel String Matching using Pattern Dividing for Intrusion Detection Systems", IEEE Transactions On Parallel And Distributed Systems, Third Draft, September 2010, pp.1-8,2011.

[42] Yi-Hua E. Yang and Viktor K. Prasanna, "Robust and Scalable String Pattern Matching for Deep Packet Inspection on Multi-core Processors", IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS, pp.1-11,2012.

[43] Yue Hu et al.," A Fast Algorithm for Multi-String Matching Based on Automata Optimization" 2nd International Conference on Future Computer and Communication,pp.379-383,Vol 2,2010

[44] Junghak Kim et al.,"Programmable Architecture for NFA-based String Matching" , pp.484-489, IEEE, 2010.

[45] Yi Tang et al.," Independent Parallel Compact Finite Automatons for Accelerating Multi-String Matching", IEEE Globecom 2010 proceedings, 2010.

[46] Xiaofei Wang, Yang Xu, Junchen Jiang, Olga Ormond, Bin Liu, and Xiaojun Wang "StriFA: Stride Finite Automata for High-Speed Regular Expression Matching in Network Intrusion Detection Systems", IEEE SYSTEMS JOURNAL, VOL. 7, NO. 3, pp. 374-384, 2013.

[47] Weirong Jiang et al.," Scalable Multi-Pipeline Architecture for High Performance Multi-Pattern String Matching",IEEE,2010.

[48] Yu Cheng and Tao Zhang, "Design of Fast Multiple String Searching Based on Improved Prefix Tree", 2010 Third International Conference on Knowledge Discovery and Data Mining, pp.111-114, 2010.

[49] Shmuel T. Klein and Dana Shapira" The String-to-Dictionary Matching Problem",2011 Data Compression Conference, pp.143-152, IEEE, 2011.

[50] KSMV Kumar, S. Viswanadha Raju and A. Govardhan, "Overlapped Text Partition Algorithm for Pattern Matching on Hypercube Networked Model", GJCST, pp. 1-8,2013.

[51] Yao Xin , Benben Liu, Biao Min, WillX.Y.Li, Ray C.C. Cheung, Anthony S.Fong, Ting Fung Chan" Parallel architecture for DNA sequence inexact matching with Burrows-Wheeler Transform", Microelectronics Journal, pp. 670-682,Elsevier,2013.

[52] Hoang Le, and Viktor K. Prasanna, "A Memory-Efficient and Modular Approach for Large-Scale String Pattern Matching", IEEE Transactions on Computers, VOL. 62, NO. 5, pp. 844-857, 2013.

[53] Jiaying Wang, Xiaochun Yang, Bin Wang, "Cache-Aware Parallel Approximate Matching and Join Algorithms Using BWT", ACM, 2013.

[54] Christoph Strecha et al.," LDAHash: Improved matching with smaller descriptors", IEEE Transactions On Pattern Analysis And Machine Intelligence, pp.1-14, 2011.

[55] Edward Fernandez et al.," String Matching in Hardware using the FM-Index**",** IEEE International Symposium on Field-Programmable Custom Computing Machines, pp.218-225, IEEE computer society, 2011

[56] TAN Jianlong et al.," Speeding up Pattern Matching by Optimal Partial String Extraction", the first international workshop on security in computers, networking and communications, pp.1030-1035, IEEE, 2011.

[57] Rajesh Prasad, Anuj Kumar Sharma, Alok Singh, Suneeta Agarwal1 and Sanjay Misra, "Efficient bit-parallel multi-patterns approximate string matching algorithms", Scientific Research and Essays Vol. 6(4), pp. 876-881, 18 February, 2011.

[58] Mosleh M. Abu-Alhaj et al.," An Innovative Platform To Improve The Performance Of Exact-String Matching ALGORITHMS",2010, (IJCSIS) International Journal of Computer Science and Information Security,pp.280-283, Vol. 7, No. 1, 2010

[59] Mosleh M. Abu-Alhaj et al.," An Innovative Platform To Improve The Performance Of Exact-String Matching ALGORITHMS",2010, (IJCSIS) International Journal of Computer Science and Information Security,pp.280-283, Vol. 7, No. 1, 2010

[60] Benedikt Forchhammer, Thorsten Papenbrock, Thomas Stening, Sven Viehmeier, Uwe Draisbach, Felix Naumann, "Duplicate Detection on GPUs", pp.165-188,2013

[61] Antonino Tumeo and G et al.," Aho-Corasick String Matching on Shared and Sistributed Memory Parallel Architectures", IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS, pp. 1-9, IEEE, 2011.

[62] Antonino Tumeo, Oreste Villa, and Daniel G. Chavarrı´a-Miranda, "Aho-Corasick String Matching on Shared and Distributed-Memory Parallel Architectures", IEEE Transactions on Parallel and Distributed Systems, VOL. 23, NO. 3, PP.436-443, 2012.

[63] Vinod.O, B.M.Sagar, "Hash-Based String Matching Algorithm For Network Intrusion Prevention systems (NIPS)", International Journal of Advanced Computer Theory and Engineering, Volume-2, Issue-2, pp.31-35, 2013.

[64] Oren Ben-Kiki, Philip Bille, Dany Breslauer, Leszek Gasieniec, Roberto Grossi, Oren Weimann "Towards Optimal Packed String Matching", pp. 1-33, ---July 10, 2013.

[65] H. D. Cheng and K. S. Fu, "VLSI Architecture for String Matching and Pattern Matching," Pattern Recognition, Vol.20, pp. 125-141, 1987.

[66] Daniel P. Lopresti, "P-NAC : A Systolic Array for Comparing Nucleic Acid Sequences," Computer, pp. 98-99, July 1987.

[67] Amar Mukhopadhyay, "Hardware Algorithms for Nonnumeric Computation," IEEE trans. on Computers, Vol. C-28, No. 6, pp. 384-394, June 1979.

[68] Bradly Fawcett, "Reconfiguring a Computing Pattern," Electronic Engineering Times, Manhasset, pp. 64- , April. 1995.

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

[70] Carla Correa Tavares dos Reis and Oswaldo Cruz, "Approximate String Matching Algorithm Using Parallel Methods for Molecular Sequence Camparisons", IEEE, pp.140-143,2005.

[71] Muhammad Zubair et al.," Text Scanning approach for Exact String Matching", International Conference on Networking and Information Technology,pp.118-121,2010.

[72] Yunho Oh , Doohwan Oh , Won W. Ro "GPU-Friendly Parallel Genome Matching with Tiled Access and Reduced State Transition Table", Springer, pp. 526-551, 2013.

[73] Tomohiro I , Shunsuke Inenaga, Masayuki Takeda "Palindrome pattern matching" Theoretical Computer Science, pp. 162-170,Elsevier,2013.

[74] Fang Xiangyan et al.," The Research and Improving for Multi-pattern String Matching Algorithm", IEEE, 2010.

[75] Zhengda Xiong "A Composite Boyer-Moore Algorithm for the String Matching Problem", The 11th International Conference on Parallel and Distributed Computing, Applications and Technologies, pp.492-496, IEEE, 2010.

[76] Golnaz Badkobeh,Gabriele Fici,Steve Kroonc,Zsuzsanna Liptákd, " Binary jumbled string matching for highly run-length compressible texts", Information Processing Letters, pp.604-608,2013.

[77] Dan Guo , Xuegang Hu , Fei Xie , XindongWu "Pattern matching with wildcards and gap-length constraints based on a centrality-degree graph" ,Springer, pp.57-74, 2013.