Global Journal of Computer Science and Technology
Software & Data Engineering
Volume 13 Issue 12 Version 1.0 Year 2013
Type: Double Blind Peer Reviewed International Research Journal
Publisher: Global Journals Inc. (USA)
Online ISSN: 0975-4172 & Print ISSN: 0975-4350

A Parallel Computational Approach for String Matching- A Novel Structure with Omega Model

By K Butchi Raju & Dr. S. Viswanadha Raju
Griet Aff to Jntuh, India

Abstract- In recent day's parallel string matching problem catch the attention of so many researchers because of the importance in different applications like IRS, Genome sequence, data cleaning etc.,. While it is very easily stated and many of the simple algorithms perform very well in practice, numerous works have been published on the subject and research is still very active. In this paper we propose a omega parallel computing model for parallel string matching. The algorithm is designed to work on omega model 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. Experimental results show that, on a multi-processor system, the omega model implementation of the proposed parallel string matching algorithm can reduce string matching time.

Keywords: *string matching; parallel string matching; computing model; omega model.*

GJCST-C Classification : *C.1.4*

Strictly as per the compliance and regulations of:

© 2013. K Butchi Raju & Dr. S. Viswanadha Raju. This is a research/review paper, distributed under the terms of the Creative Commons Attribution-Noncommercial 3.0 Unported License http://creativecommons.org/licenses/by-nc/3.0/), permitting all non-commercial use, distribution, and reproduction inany medium, provided the original work is properly cited.

A Parallel Computational Approach for String Matching- A Novel Structure with Omega Model

K Butchi Raju [α] & Dr. S. Viswanadha Raju [σ]

Abstract- In recent day's parallel string matching problem catch the attention of so many researchers because of the importance in different applications like IRS, Genome sequence, data cleaning etc.,. While it is very easily stated and many of the simple algorithms perform very well in practice, numerous works have been published on the subject and research is still very active. In this paper we propose a omega parallel computing model for parallel string matching. The algorithm is designed to work on omega model 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. Experimental results show that, on a multi-processor system, the omega model implementation of the proposed parallel string matching algorithm can reduce string matching time.

Keywords: string matching; parallel string matching; computing model; omega model.

I. 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 algorit- hms, algorithmic models and finally conclude the study.

Author α: Associate Professor, Department of CSE, GRIET, Hyderabad, AP, India.
Author σ: Professor & HOD, Department of CSE, JNTU College of Engineering, JNTUH, Jagithyal, A.P., India.

II. 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].

III. Literature

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,

© 2013 Global Journals Inc. (US)

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.

a) Intrusion Detection Systems (Ids)

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.

Dharmapurikar et al.[13] 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., [14] 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.,[15] 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 the above papers, it is possible to improve Snort’s efficiency using distributed environment and testability of Snort has been enhanced.

b) Dfa Based Approaches

Dharmapurikar et al. presented a scheme[16] 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 [17]. 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. [18] 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.[19] 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 [20] introduced the sampling techniques to accelerate regex matching, but not all kinds of regex are supported.

WANG Xiaofei et. al[21] 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 [22] 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

© 2013 Global Journals Inc. (US)

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.

c) Parallel Processing Based Approaches

Akhtar Rasool and Nilay Khare[23] 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[24] 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[25] 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.

d) Aho Corasick Based Approaches

Wei Lin, Bin Liu[26] 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[27] 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.

Hyun Jin Kim et.al [28] 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

© 2013 Global Journals Inc. (US)

mapped onto a string matcher. In the evaluations of Snort rules, the proposed string matching outperforms the existing bit-split string matching.

Alicherry et al. [29] 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. [30] and W. Lin and B. Liu [31] 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. [32] 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. [33] 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. [34] 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.

e) Finite Automata Based Approaches

HyunJin Kim et al.,[35] 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 [36] 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.

Yi Tang et al., [37] 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.

f) Hardware Related Based Approaches

KSMV Kumar et.al [38] 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 [39] presented a hardware architecture for the BWT-based inexact sequence mapping algorithm using the Field Programmable Gate

© 2013 Global Journals Inc. (US)

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[39].

Hoang Le, and Viktor K. Prasanna [40] 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.

TAN Jianlong et al., [41] 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.,[42] 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). Mosleh M. Abu-Alhaj et al.,[43] 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

© 2013 Global Journals Inc. (US)

performance of the PXSMAlg platform over the sequential methods

g) Gpu's Based Approaches

Benedikt Forchhammeret.al [44] 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.,[45] 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 [46] 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.

h) Ram Based Approaches

Vinod.O et.al [47] 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.,[48] 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.

© 2013 Global Journals Inc. (US)

i) 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 [49] proposed VLSI architecture of two dimensional arrangements of n*m processing elements. P-NAC (Princeton Nucleic Acid Comparator) [50] was built using linear systolic array architecture for comparing DNA sequences.

Similarly, Sastry et al [51] 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 [52]. As Foster and Kung [53] 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[54] 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.,[55] 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.

Tomohiro I et al.,[56] 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 space between the left and right arms of the palindromes.

IV. Scattered Computing and Scattered Algorithms

The new technologies of networking and the dramatic evolution of the internet and intranet impacts the way that we use computers and changes the way we create applications for them. Distributed applications are becoming the natural way to build software. “Distributed computing” is all about designing and building applications as a set of processes that are distributed across a network of machines and work together as an ensemble to solve a common problem[8-11]. Distributed algorithms are algorithms designed to run on a distributed system; where many processes cooperate by solving parts of a given problem in parallel. For this purpose, the processes have to exchange data and synchronize their actions. In contrast to so called parallel algorithms, communication and

© 2013 Global Journals Inc. (US)

synchronization is solely done by message passing - there are no shared variables- and usually the processes do not even have access to a common clock. Since message transmission time cannot be ignored, no process has immediate access to the global state. Hence, control decisions must be made on a partial and often outdated view of the global state which is assembled from information gathered gradually from other processes. Distributing computing requires a tool by which the distributed machines can communicate. Many tools are available such as Remote Method Invocation (RMI), CORBA and Java Space. Each tool has its own specifications; the application designer chooses the appropriate one for his application requirement[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 and proposed a computing model with omega architecture.

V. Proposed System Architecture

a) System Architecture

System Architecture describes "the overall structure of the system and the ways in which the structure provides conceptual integrity". Architecture is the hierarchical structure of a program components (modules), the manner in which these components interact and the structure of data that are used by that components. The existing string matching system architecture is as shown in Fig 1 and in this the efficiency is not good.

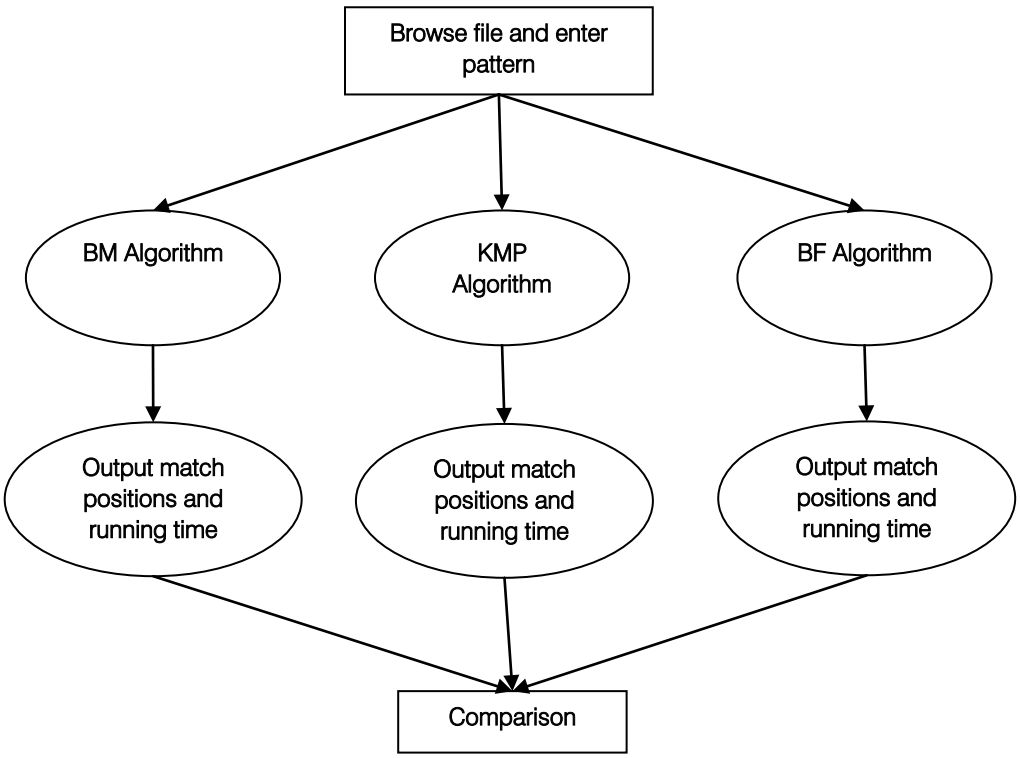

Figure 1 : Existing System

In the existing string matching architecture we search the required pattern sequentially at first we pass the required that is to be searched and this pattern is searched by using the three algorithms Brute force, KMP, Boyer Moore the entire string is passed through all the algorithms and the output match and the running time is calculate for the required pattern from all the algorithms and the algorithm with the least running time is selected, all this is done sequentially which takes more time to execute to improve the efficiency and the performance in this we use the parallel string matching algorithms with multicores processors as shown in Fig2.

The proposed system Architecture of Comparison of parallel String Matching Algorithms is as follows in the below diagram. In this search the pattern parallel. in this at first we take the input as a string or text. The required text that is to be searched is further divided into further small patterns and all this patterns are passed on the different parallel algorithms like KMP boyar Moore, brute force and at all the output position match and running time of all the patterns is calculated and the all the patterns of same algorithm are added and all the resulted running time are compared with other algorithms resulting time and from them the best one is taken as the efficient algorithm for the string matching.

© 2013 Global Journals Inc. (US)

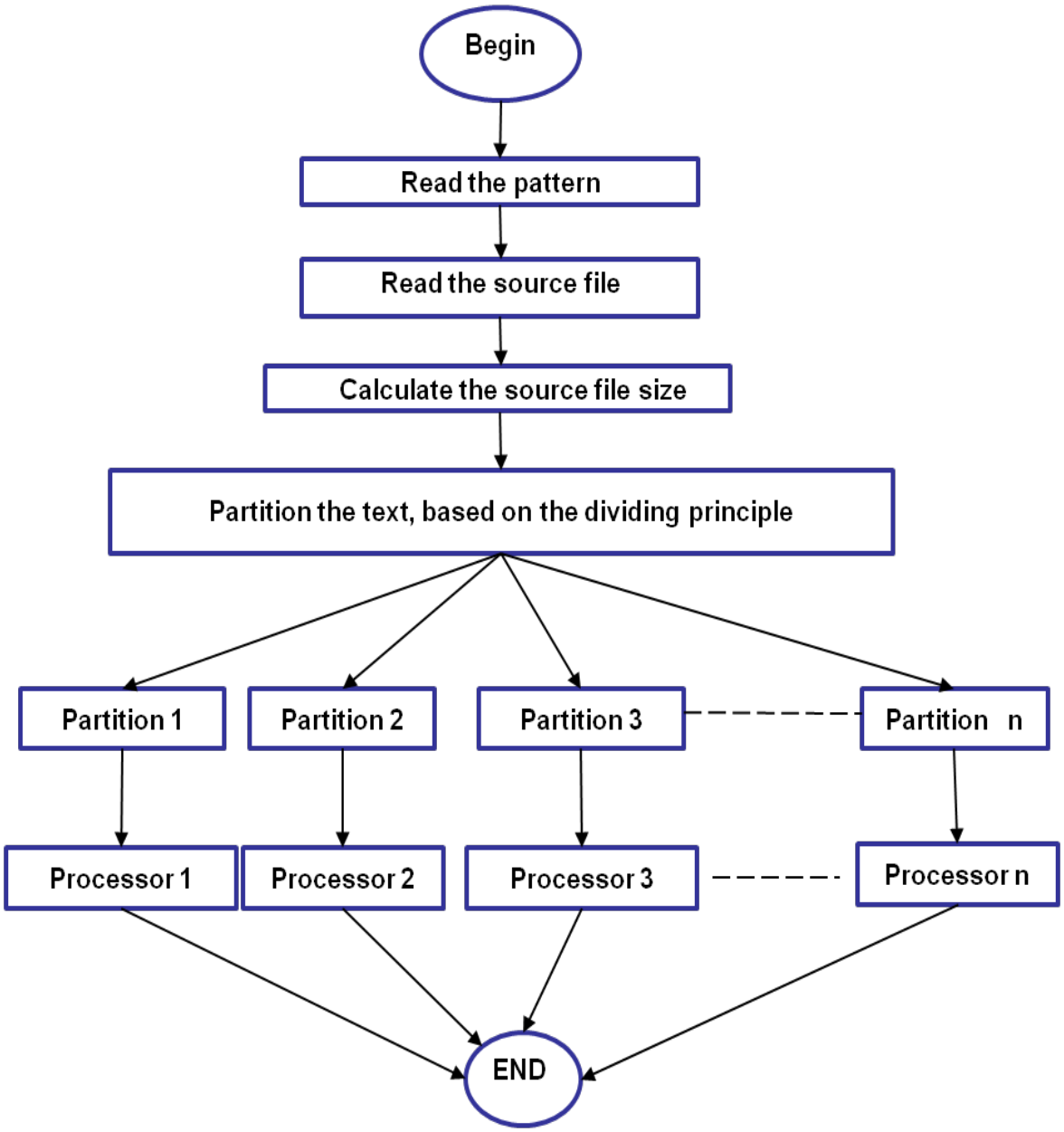

Figure 2 : Proposed Systems

b) Proposed Approach

In now a day as the current free textual database is growing vast there is a problem of finding the pattern by string matching the efficiency is decreased and takes more time. In our paper, we use parallel algorithms to increase the efficiency on multicore processor we pass the same string to all the three algorithms and we select the best based on the running time. Here we have to implement the proposed system with JAVA 1.7 multi threading; initially we have to implement the proposed algorithm with threading on Multicore processor. Here we discuss some of them.

© 2013 Global Journals Inc. (US)

1. Choose the TEXT file and Open the file in read mode
2. Read a line from TEXTFILE into variable ProcessLine (text of size n)
3. Read the pattern of size m and number of processors
4. Distribute the pattern to all the processors in the omega model structure
5. Each Processor searches the pattern in the ProcessLine with proposed approach and returns the result

Proposed approach

Begin

5.1. If ProcessLine equals to NULL GOTO Step 5.7
5.2. It compares the last character of the pattern with the rightmost text character of the ProcessLine, then
5.3. if found it compares the first character of the pattern with the leftmost text character of the window, then
5.4. if found it compares the middle character of the pattern with the middle text character of the window. And finally
5.5. if found it actually compares the other characters from the second to the last but one.
else
5.6. If a mismatch occurs, it will shift according to the value in pre-processing stage.
5.7. Increment the counter in TEXTFILE and GOTO Step 5.1. Repeat the steps from 5.1 to 5.6 until the end of the file.

End

Pre-Processing Stage

Computing model consists of the good suffix and bad-character shift function. The bad character shift means to shift the pattern so that the text character of the mismatch is aligned to the last occurrence of that character in the initial part of the pattern (pattern minus last pattern character), if there is such an occurrence, or one position before the pattern if the mismatched character doesn't appear in the initial part of the pattern at all.

The other shift, the good suffix shift, aligns the matched part of the text, m, with the rightmost occurrence of that character sequence in the pattern that is preceded by a different character (including none, if the matched suffix is also a prefix of the pattern) than the matched suffix m of the pattern.

6. Each processor stores the MATCH results and RETURNS to the main program.
7. The main program collect the return results from all the processors and summing them
8. STOP.

c) Claims

Implementation is the stage where the theoretical design is turned into a working system. The most crucial stage in achieving a new successful system and in giving confidence on the system for the users that will work efficiently and effectively. The system will be implemented only after thorough testing and if it is found to work according to the specification. For testing our proposed system we will take the gene sequence data set, consists of the four nucleotides a, c, g and t (standing for adenine, cytosine, guanine, and thymine, respectively) used to encode DNA. Therefore, the alphabet is O={A, C, G, T}. The text is consisted of 7,50,000 records. Our test tested with different processors like i3, i5 etc., here we put some achievements what we develop and observe, finally our system shows that parallel approach is much better than sequential approach with multi core processor. The Fig 3 shows (Graph) Execution time vs File size on sequential search with intel i3 processor using Brute force algorithm. From the graph we clearly observe that sequential is better compared to parallel approach. The Fig 4 shows (Graph) Execution time vs File size on sequential search with intel i5 processor using Brute force algorithm. From the graph we clearly observe that sequential is better compared to parallel approach.

© 2013 Global Journals Inc. (US)

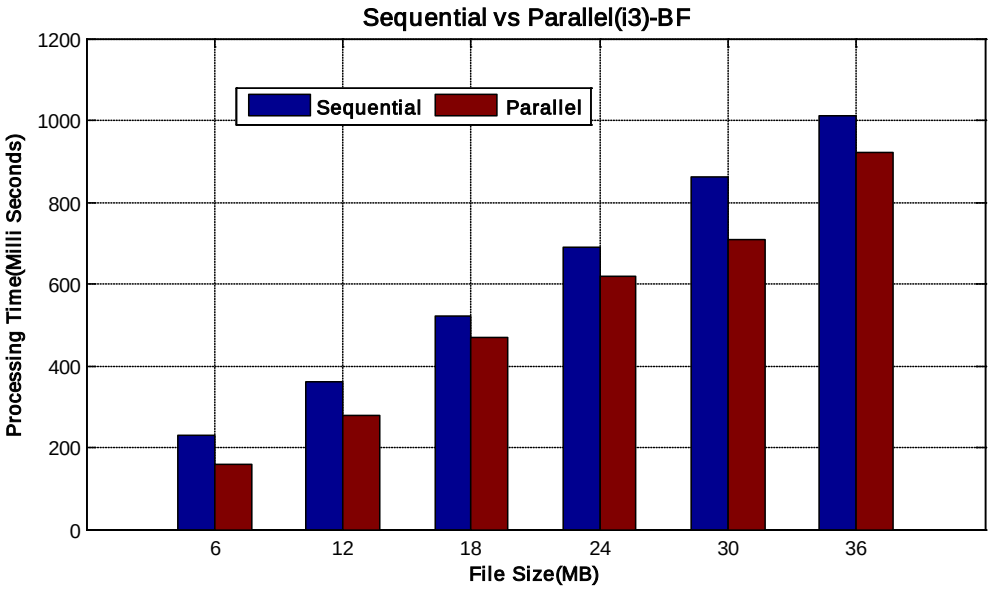

Figure 3 : Sequential VS Parallel(i3)-BF

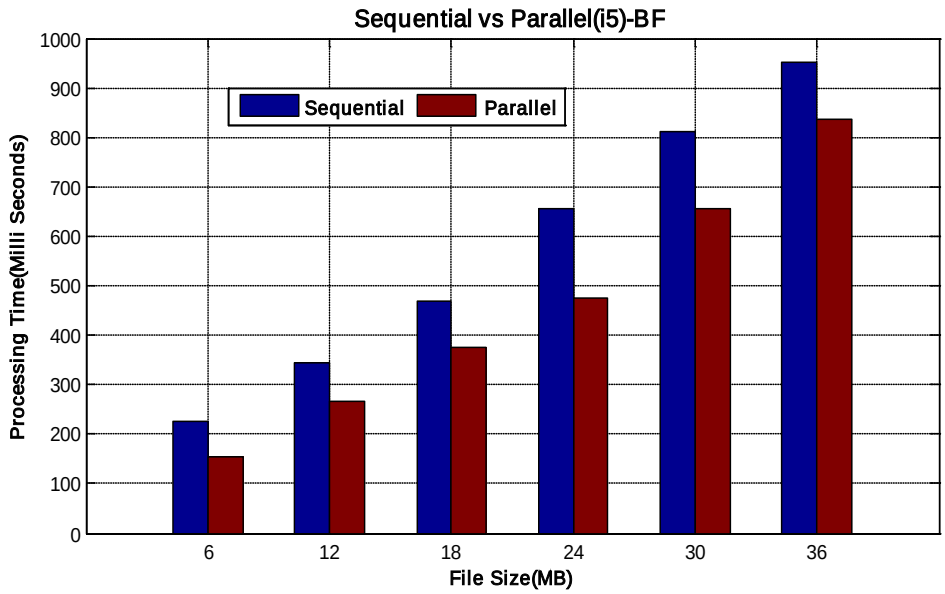

Figure 4 : Sequential VS Parallel(i5)-BF

The Fig 5 shows(Graph) Execution time vs File size on sequential search with intel i3 processor using KMP algorithm. From the graph we clearly observe that sequential is better compared to parallel approach. The Fig 6 shows(Graph) Execution time vs File size on sequential search with intel i5 processor using KMP algorithm. From the graph we clearly observe that sequential is better compared to parallel approach.

© 2013 Global Journals Inc. (US)

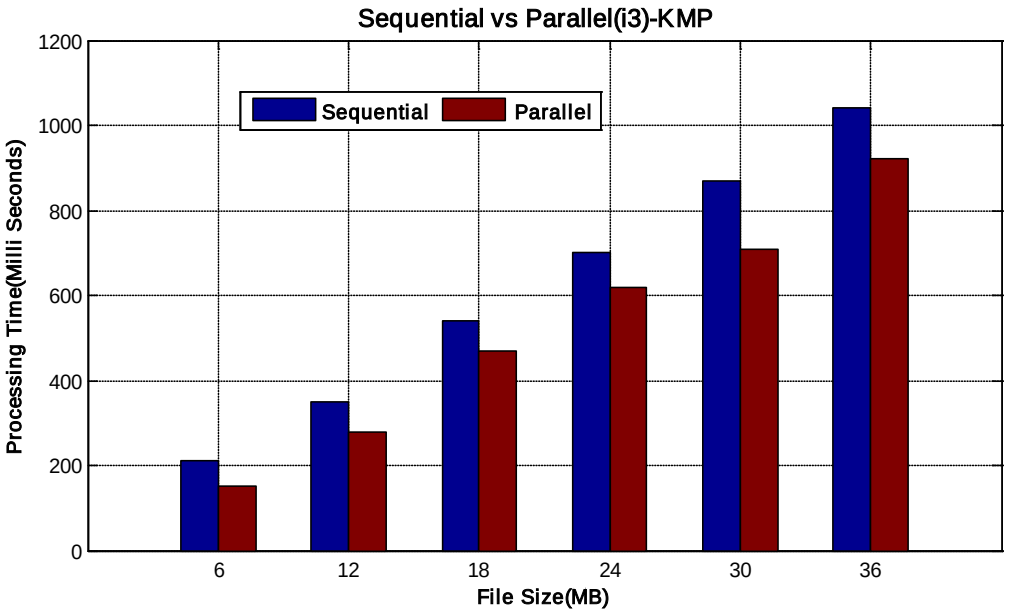

Figure 5 : Sequential VS Parallel(i3)-KMP

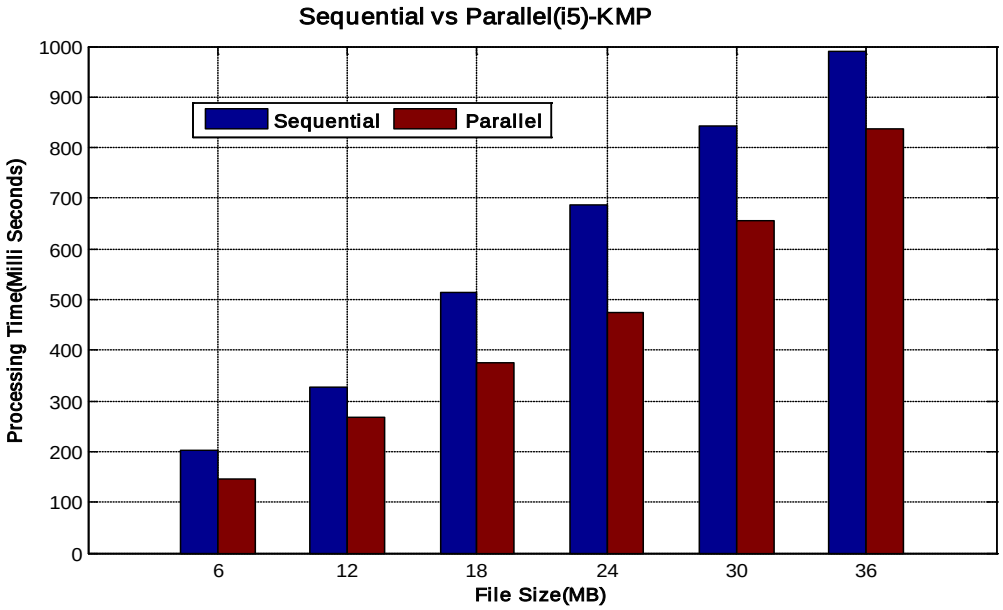

Figure 6 : Sequential VS Parallel(i5)-KMP

The Fig 7 shows (Graph) Execution time vs File size on sequential search with intel i3 processor using BM algorithm. From the graph we clearly observe that sequential is better compared to parallel approach. The Fig 8shows(Graph) Execution time vs File size on sequential search with intel i5 processor using BM algorithm. From the graph we clearly observe that sequential is better compared to parallel approach.

© 2013 Global Journals Inc. (US)

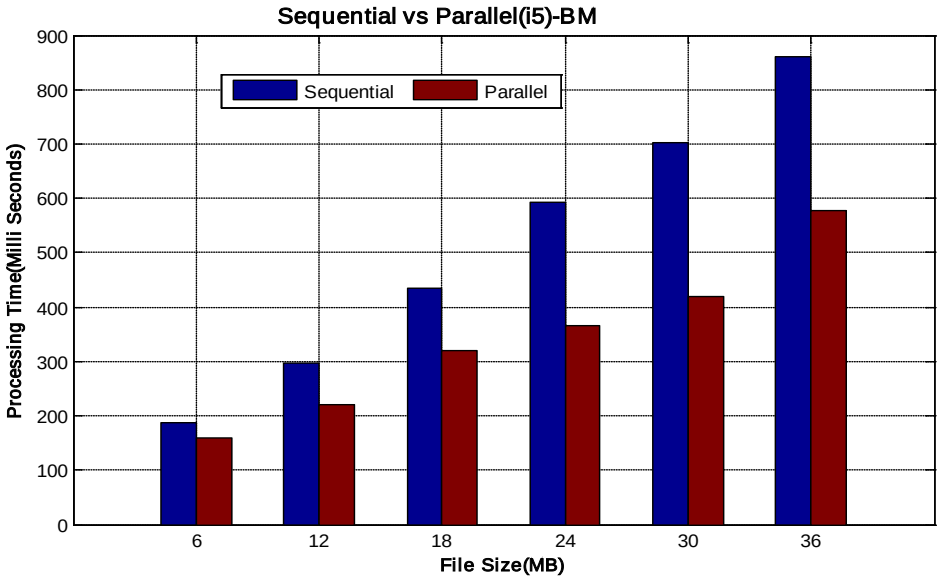

Figure 7 : Sequential VS Parallel(i5)-BM

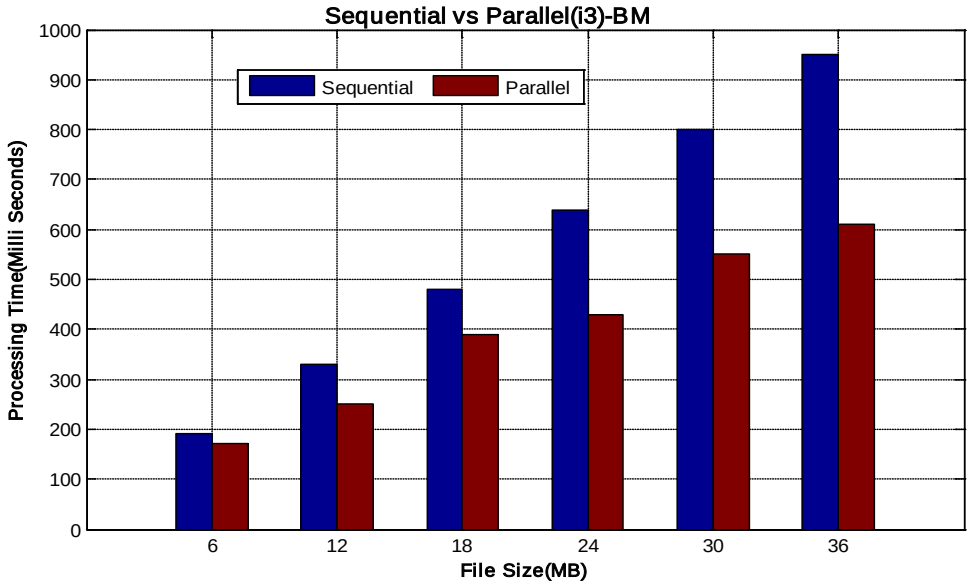

Figure 8 : Sequential VS Parallel(i3)-BM

The Fig 9 shows (Graph) Execution time vs File size on sequential search with intel i3 processor using Proposed algorithm. From the graph we clearly observe that sequential is better compared to parallel approach. The Fig 10 shows(Graph) Execution time vs File size on sequential search with intel i5 processor using Proposed algorithm. From the graph we clearly observe that sequential is better compared to parallel approach.

© 2013 Global Journals Inc. (US)

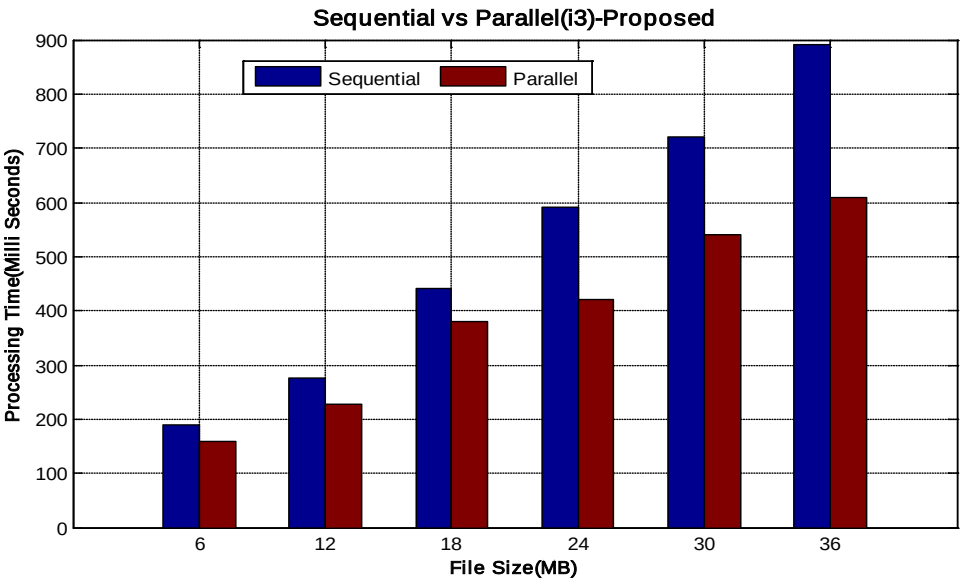

Figure 9 : Sequential VS Parallel(i3)-Proposed

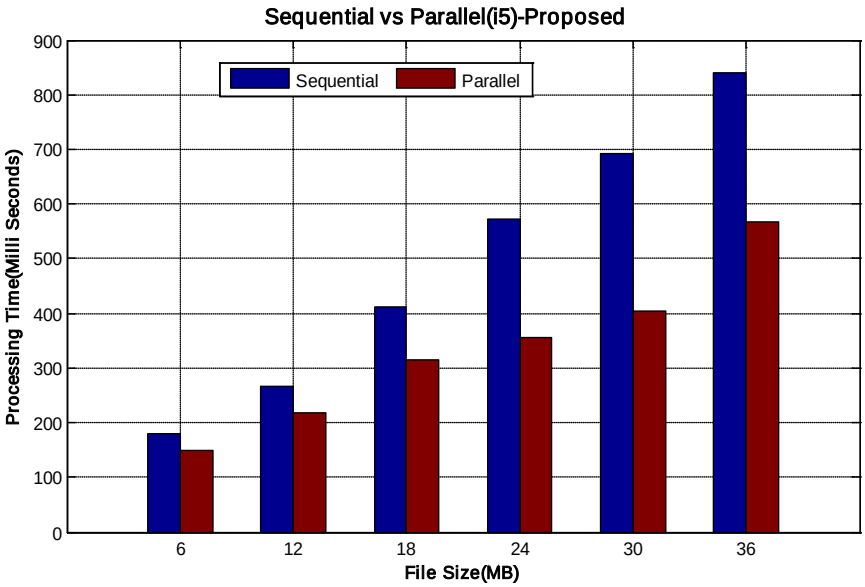

Figure 10 : Sequential VS Parallel(i5)-Proposed

The Fig 11 shows (Graph) Execution time vs File size on sequential search with intel i3 processor using Boyer Moore, Brute force, KMP and proposed Algorithm. This graph shows the performance difference between Boyer Moore, Knuth Morris Pratt and Brute force algorithms. From the graph clearly observe that proposed is better compared to other approaches.

© 2013 Global Journals Inc. (US)

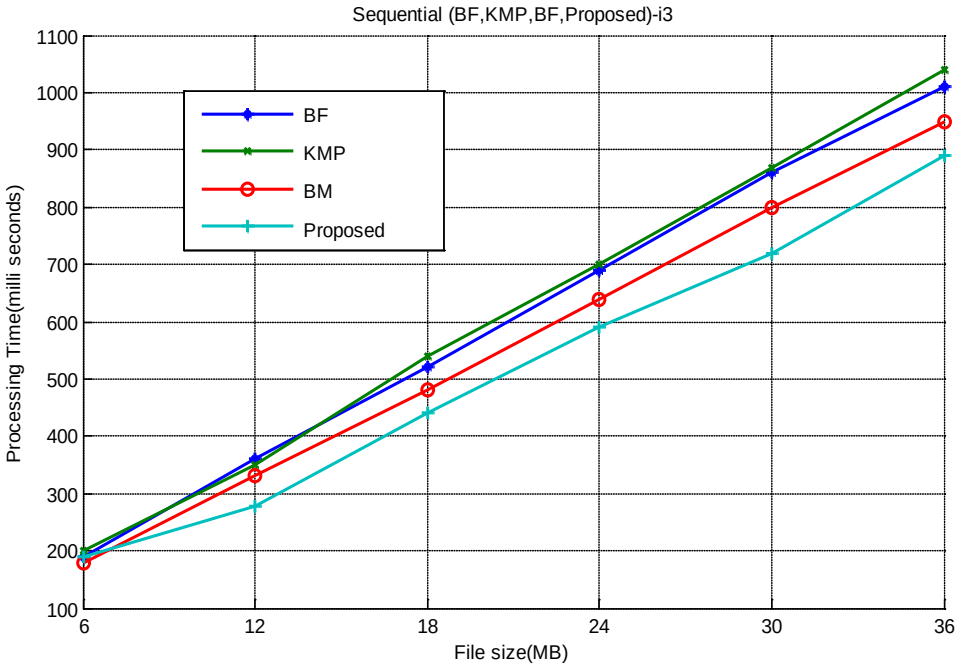

Figure 11 : Sequential approaches BF,BM,KMP and Proposed (i3)

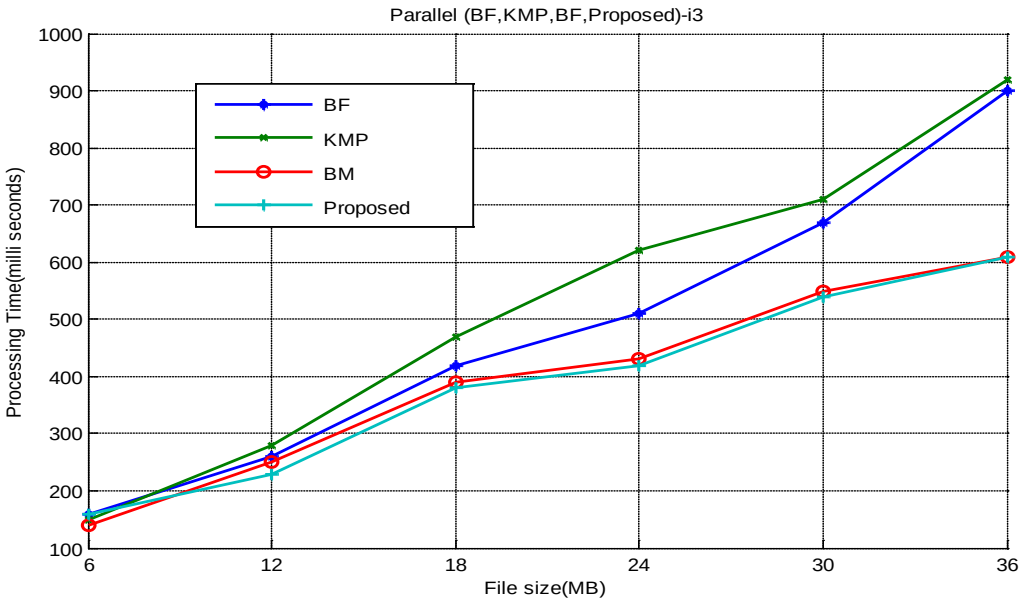

Figure 12 : parallel approaches BF,BM,KMP and Proposed (i3)

The Fig 12 shows (Graph) Execution time vs File size on parallel search with intel i3 processor using Boyer Moore, Brute force, KMP and proposed Algorithm. This graph shows the performance difference between Boyer Moore, Knuth Morris Pratt and Brute force algorithms. From the graph clearly observe that proposed is better compared to other approaches, as well as this parallel approach is much better compared to sequential approaches.

The Fig 13 shows (Graph) Execution time vs File size on sequential search with intel i5 processor using Boyer Moore, Brute force, KMP and proposed Algorithm. This graph shows the performance difference between Boyer Moore, Knuth Morris Pratt and Brute force algorithms. From the graph clearly observe that proposed is better compared to other approaches.

© 2013 Global Journals Inc. (US)

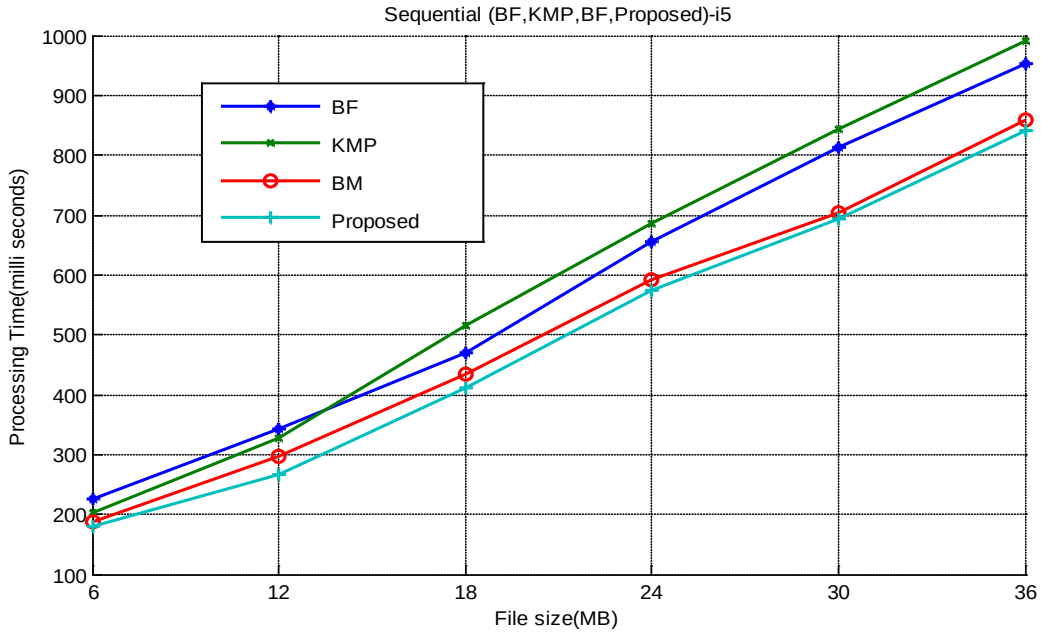

Figure 13 : sequential approaches BF, BM, KMP and Proposed (i5)

The Fig 14 shows (Graph) Execution time vs File size on parallel search with intel i5 processor using Boyer Moore, Brute force, KMP and proposed Algorithm. This graph shows the performance difference between Boyer Moore, Knuth Morris Pratt and Brute force algorithms. From the graph clearly observe that proposed is better compared to other approaches, as well as this parallel approach is much better compared to sequential approaches.

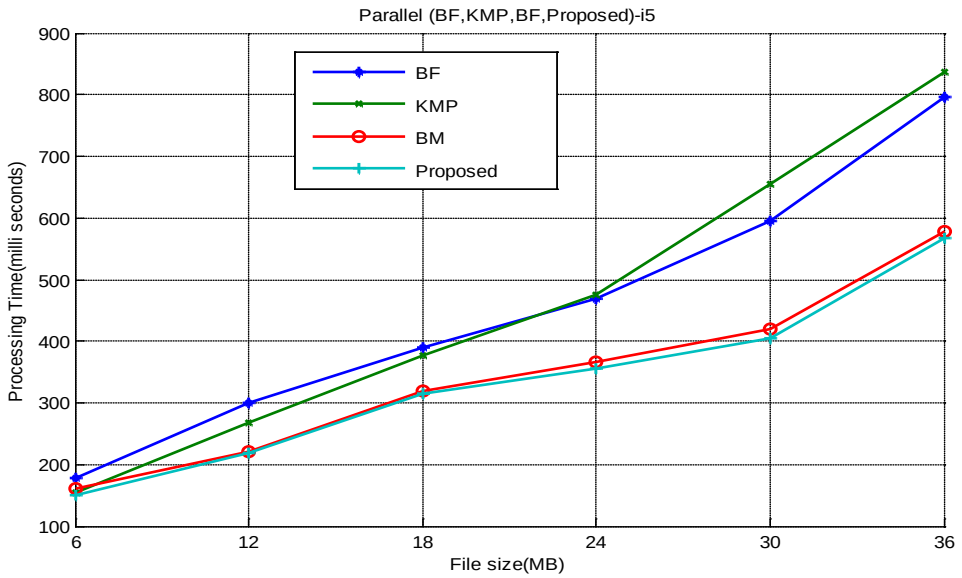

Figure 13 : Parallel approaches BF,BM,KMP and Proposed (i5)

VI. Conclusions

In this study, we widely investigate the problem of parallel string matching in the context of Parallel Computing with omega model. This Parallelization greatly improves the matching efficiency if the text size is very large and a sufficient numbers of processors are available. The most important characteristic of the proposed algorithm is that reduces the number of comparisons by making better use of next bit characters. Further parallelization of proposed algorithm provides parallel computing of pattern searching on the text in omega architecture. Experimental results shows that proposed algorithm much more better than other approaches both sequential and parallel.

© 2013 Global Journals Inc. (US)

References Références Referencias

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. 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.
14. 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
15. Safaa O. Al-Mamory et al.," String Matching Enhancement for Snort IDS" , pp.1020-1023.
16. 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.
17. 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.
18. 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.
19. 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.
20. 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.
21. 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.
22. 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.
23. 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.
24. 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.
25. Panwei Cao and Suping Wu , "Parallel Research on KMP Algorithm", pp.4252-4255, IEEE, 2011.
26. 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.

© 2013 Global Journals Inc. (US)

27. Chuanpeng Chen and Zhongping Qin A Bit-split Byte-parallel String Matching Architecture, IEEE ,pp.214- 217, 2009.
28. 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.
29. M. Alicherry, M. Muthuprasanna and V. Kumar, High speed pattern matching for network IDS/IPS, *IEEE ICNP*, pp.187-196, 2006.
30. 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.
31. W. Lin and B. Liu, Pipelined parallel AC-based approach for multi-string matching, *IEEE ICPADS*, pp.665-672, 2008.
32. N. Hua, H. Song and T. V. Lakshman, Variable-stride multi-pattern matching for scalable deep packet inspection, *IEEE INFOCOM*, pp.415-423, 2009.
33. D. P. Scarpazza, O. Villa and F. Petrini, Exact multi-pattern string matching on the cell/b.e. processor, *ACM CF*, 2008.
34. 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.
35. 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.
36. 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.
37. Yi Tang et al.," Independent Parallel Compact Finite Automatons for Accelerating Multi-String Matching", IEEE Globecom 2010 proceedings, 2010.
38. KSMV Kumar, S. Viswanadha Raju and A. Govardhan, "Overlapped Text Partition Algorithm for Pattern Matching on Hypercube Networked Model", GJCST, pp. 1-8,2013.
39. 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.
40. 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.
41. 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.
42. 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.
43. 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
44. Benedikt Forchhammer, Thorsten Papenbrock, Thomas Stening, Sven Viehmeier, Uwe Draisbach, Felix Naumann, "Duplicate Detection on GPUs", pp.165-188,2013
45. 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.
46. 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.
47. 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.
48. Oren Ben-Kiki, Philip Bille, Dany Breslauer, Leszek Gasieniec, Roberto Grossi, Oren Weimann "Towards Optimal Packed String Matching", pp. 1-33, ---July 10, 2013.
49. H. D. Cheng and K. S. Fu, "VLSI Architecture for String Matching and Pattern Matching," Pattern Recognition, Vol.20, pp. 125-141, 1987.
50. Daniel P. Lopresti, "P-NAC : A Systolic Array for Comparing Nucleic Acid Sequences," Computer, pp. 98- 99, July 1987.
51. Amar Mukhopadhyay, "Hardware Algorithms for Nonnumeric Computation," IEEE trans. on Computers, Vol. C-28, No. 6, pp. 384-394, June 1979.
52. Bradly Fawcett, "Reconfiguring a Computing Pattern," Electronic Engineering Times, Manhasset, pp. 64- , April. 1995.
53. M. J. Foster and H. T. Kung, "The Design of Special- Purpose VLSI Chips," Computer, pp. 26-38, Jan. 1980.
54. Carla Correa Tavares dos Reis and Oswaldo Cruz, "Approximate String Matching Algorithm Using Parallel Methods for Molecular Sequence Camparisons", IEEE, pp.140-143,2005.

© 2013 Global Journals Inc. (US)

55. Muhammad Zubair et al.," Text Scanning approach for Exact String Matching", International Conference on Networking and Information Technology,pp.118-121,2010.
56. Tomohiro I , Shunsuke Inenaga, Masayuki Takeda "Palindrome pattern matching" Theoretical Computer Science, pp. 162-170,Elsevier,2013.

© 2013 Global Journals Inc. (US)

This page is intentionally left blank

© 2013 Global Journals Inc. (US)