Global Journal of Computer Science and Technology
Network, Web & Security
Volume 13 Issue 4 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

# Overlapped Text Partition Algorithm for Pattern Matching on Hypercube Networked Model

By Prof. KSMV Kumar, Prof. S. Viswanadha Raju
& Prof. A. Govardhan
*Jawaharlal Nehru Technological University*

*Abstract -* The web has been continuously growing and getting hourglass shape. The indexed web is measured to contain at least 30 billion pages. It is no surprise that searching data poses serious challenges in terms of quality and speed. Another important subtask of the pattern discovery process is sting matching, where in which the pattern occurrence is already known and we need determine how often and where it is occurs in given text. The target of current research challenges and identified the new trends i.e distributed environment where in which the given text file is divided into subparts and distributed to N no. of processors organized in hypercube networked fashion .To improve the search speed and reduce the time complexity we need to run the string matching algorithms in parallel distributed environment called as hypercube networked model using RMI method. we 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.

*Keywords :* *indexed web, pattern, text, distributed, hypercube network, RMI method and string matching.*

*GJCST-E Classification :* *C.2.5*


*Strictly as per the compliance and regulations of:*

# Overlapped Text Partition Algorithm for Pattern Matching on Hypercube Networked Model

Prof. KSMV Kumar [α], Prof. S. Viswanadha Raju [σ] & Prof. A. Govardhan [ρ]

*Abstract -* The web has been continuously growing and getting hourglass shape. The indexed web is measured to contain at least 30 billion pages. It is no surprise that searching data poses serious challenges in terms of quality and speed. Another important subtask of the pattern discovery process is sting matching, where in which the pattern occurrence is already known and we need determine how often and where it is occurs in given text. The target of current research challenges and identified the new trends i.e distributed environment where in which the given text file is divided into subparts and distributed to N no. of processors organized in hypercube networked fashion .To improve the search speed and reduce the time complexity we need to run the string matching algorithms in parallel distributed environment called as hypercube networked model using RMI method. we 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.

*Keywords : indexed web, pattern, text, distributed, hypercube network, RMI method and string matching.*

## I. Introduction

String matching diversely used in many areas of computer sciences. It has been one of the prominent issues of information retrieval system. Some standard algorithms have been used for processing text files against patterns, for example in manipulation of text, text compression, network analysis and also in data retrieval systems. The algorithms studied in the present character forms the basic components in its software implementation and also serve as a model in fields of computer science like system design purposes, web search engines, computer virus signature matching and networking [1]. Rapid growth of abundant information makes necessary to have efficient methods for information retrieval. Coping with the growth of the web and query traffic requires scalable information retrieval systems. Today commercial search engines are fully automatic and their web index on a few data centers [2]. It is tedious task to come up with scalable indexing and query processing techniques for next generation IRS in the coming future. The web comprises wide variety of content in the form of structured Meta data, databases, maps, images, videos and textual documents etc. [1]. The main challenge of present IRS may be scalability. A recent trends envisions that the number of servers required by a search engines to keep up with the load in 2010 may be in the order of millions as such the text size is increasing to tens of billions of pages [1,2]. Hence it is very urgent to design a truly distributed large scale systems that enables fast and accurate search over very large amount of content [15]. In this paper we mainly focus on pattern matching on distributed environment called as hypercube network model using RMI method [14]. Given a pattern may be more common or more specific, we wish to count how many times it occurs in the text and to point out its occurrence positions. For pattern matching we used Kumar Viswanadha-KMP and Kumar Viswanadha- Boyer Moore string matching algorithms for different text files against different pattern files [2, 3]. Basically Boyer -Moore algorithm is works based on two heuristics: bad character heuristics and good suffix heuristics. The text files is partitioned and processed in two ways, one is non-overlapping and second is overlapping text partitioned processing [2]. In both the cases KV-KMP and KV-BM are applied for string matching (pattern matching) and the remote server will be invoked using JAVA RMI method on hypercube networked model to reduce the search time.

The paper is organized as follows section II deals with literature survey of string matching in parallel environment, section III deals with text processing techniques called as overlapping and non-overlapping text partitioned in divide and conquer paradigm. Section IV explains about the hypercube model networked systems. Section V presents experimental setup. Result analysis and discussions were discussed in section VI and VII is conclusion. Section VIII gives the references.

## II. Literaure Survey

Large amounts of data and textual information has been continuously increasing in many fields of information systems. Rapid growth of abundant information makes necessary to have efficient methods for information retrieval [1]. This chapter will give an idea how far the algorithms have helped in achieving the desired information along with its time complexities. String matching scan be accomplished by designing algorithms in two categories namely, *exact string*

*Author α : KSMV KUMAR, Association Prof in CSE Dept. SIET, IBP, AP, India-501506. E-mail : ksmvkumar@yahoo.co.in*
*Author σ : S. Viswanadha Raju, Professor in CSE Dept. JNTUH, Jagityal, AP, India-500000. E-mail : svihere@here.com*
*Author ρ : A. Govardhan professor in CSE Dept. JNTUH, kukatpally, HYD. AP, India - 500085. E-mail : agovarancse@yahoo.com*

*matching* algorithms that locates exact match of the pattern in the text string or source string and *approximate string matching* algorithms that finds closest possible match of pattern in the text with some mismatches. Exact string matching problem can be addressed in two ways *software based approach* or *hardware based approach*. Software based algorithms are slow on comparison with hardware based [22]. Hardware based solutions to string matching provides efficient data storage and fast matching [12]. String matching application in network intrusion detection systems require that the matching shall be accomplished at wire speed , software based solutions could not afford this, due to which hardware based algorithms are chosen mostly over software grounded algorithms. The algorithms discussed below addresses mostly exact string matching. Text represents an important form of data involving a lot of operations [6]. Pattern matching is one of the problems encountered in text manipulation. It is about searching and locating substring within a sequence of characters in a raw text.

*a) Software Based String Matching Algorithms*

In 1972, Cook exhibited string matching using two way push down auto meta and solved pattern matching in O(m+n) time in worst case where m and n are the lengths of text and pattern respectively[19]. However in 1977 Rivest determined that every string matching algorithm must go through at least n-m+1 comparison at worst. This shows there is no solution of obtaining a sub linear n worst time in solving the issue. It means the time needed to run an algorithm must be a function of its input size. The next algorithm discussed makes an attempt to achieve a sub-linear matching time.

Donald Knuth-Voughan Pratt-James H. Morrris (1977) basing on modifications of Cook's theorem came up with a new string matching algorithm popularly known as KMP Algorithm, briefly discussed below [3]. It is the first linear pattern matching algorithm discovered with a run time of O(m+n).

i. *Knuth-Morris-Pratt Algorithm*

Knuth Morris Pratt's string matching algorithm employs exact string matching technique with linear time complexity [3]. It involves pattern preprocessing .It scans the text string from left to right for pattern matching, while scanning the text it stores the information about the matched characters and whenever a mismatch occurs it uses this information to avoid unnecessary comparisons by sometimes shifting more than one position. It thus avoids backtracking and reduces the number of comparisons unlike naïve approach which wastes the scan information. The present algorithm uses a sliding window which slides over text string and makes shifts as per mismatches. It smartly shifts the pattern over text than the brute-force approach. Window shift uses a KMP formulated prefix function obtained by preprocessing pattern to reduce unnecessary comparisons. The algorithm uses this function to decide about the number of characters to be skipped while shifting the window whenever a mismatch takes place.

ii. *Aho-Corasick Algorithm*

Unix fgrep command implementation is based on Aho-Corasick algorithm which locates finite and fixed set of strings in a file and outputs the lines containing at least one of the strings [8]. Consider a dictionary (X) containing a fixed set of strings and a text denoted by Y. Let k be the number of strings present in X. Suppose if we wish to find all the occurrences of all the strings of a dictionary. The simple solution would be to repeatedly implement few string matching algorithms on each string .The time complexity of this operation will be O(m+n*k), where m is the sum of the lengths of the k strings of dictionary X and n is the length of the text Y. This indicates the inefficiency of this approach as the text has to be read for k times. This problem is addressed by Aho-Corasick algorithm discussed below, it undergoes sequential read of the text and run time would be O(m+n).The present algorithm extends the weaker versions of Knuth-Morris-Pratt algorithm and also fastly matches a number of patterns at one time against a single text [3].

This algorithm locates all the occurrences of finite number of keywords in a string of text. It involves construction of a finite state pattern matching machine, an automaton and then uses the machine for text processing in a single. A keyword is a finite set of strings denoted by $K= y_1, y_2,...y_n$ and let $X$ be an arbitrary text string. Pattern matching machine employs three functions namely, goto function *g*, failure function *f*, and output function *output.*

iii. *Boyer Moore Algorithm*

*B*ob Boyer and J. Strother Moore discovered this algorithm in the year 1977 which is known as one of the most efficient algorithms and also stands as a benchmark for string matching process [6]. The algorithm compares pattern string within a sliding window over a text string, employing right to left scan of characters inside the window where as the window slides from left to right over the text. The aim of this algorithm is to avoid certain fragments of text that are not eligible for comparison. This decision is taken by placing the window in left alignment with text. The algorithm starts comparing the pattern characters with the text characters in the order of right to left. If 'm' being the length of pattern (x), the algorithm compares $x_m=y_m$, where 'y' symbolizes text. On true result of this comparison the procedure continues with $x_{m-1}=y_{m-1}$ and on the occurrence of false, the algorithm makes two ways out. One is named as bad character shift or occurrence shift and the other is called as good suffix shift or better factor shift or sometimes matching shift.

On grounds of these two measures the window makes shifts and locates the pattern. These measures are explained in the following paragraphs with an example demonstration.

iv. *Horspool Algorithm*

Boyer Moore algorithm uses two gauges to know shift distance. Good suffix shift is quite complicate to implement so there was a need of a simplified algorithm using bad character measure [7]. This algorithm is a simplification of Boyer –Moore algorithm based on bad character shift. It has been produced by Nigel Horspool in the year 1980.The reason for this simplification is pattern is not always periodic. The concept used is when a bad character, reason for a mismatch is encountered; the shift decision is made by analyzing the characters towards the right of the text window.

a. *Working Principle*

The process starts by a window on text string of size equal to pattern. The scan of elements goes through right to left inside the window whereas the window slides from left to right over the text.

When a mismatch is countered for some wt[i] != wp[j], $0 \leq (i, j) < m$, wt → text window character and wp → pattern window character. Then the match of the right most character of the text window is looked in the pattern so that wt [m-1] = wt[i], where $0 \leq i < m-1$, when both found the characters are therefore aligned causing a window shift.

Case I: Suppose the bad character does not exist in the pattern then shift the whole window of size pattern.

Case II: There exists two matches of the bad character in the pattern then the rightmost character is preferred.

*b) Hardware Based String Matching Algorithms*

i. *Mishina Algorithm*

Mishina produced a string matching algorithm for vector processors in the year 1993.This algorithm is used by Hitachi's pipelined vector processor and Integrated vector processor. A vector processor also known as an array processor is a CPU which executes instructions in a single dimensional array of data items [20]. Meaning it can perform parallel computations on the elements of array. The current algorithm works in two phases: *cutout* and *check.* In the first phase, that is in cutout segment the text string is divided into autonomous serviceable substrings so that each substring can be tested for equality with respective pattern strings in a pipeline using array processors. In the next phase, as the name indicates check phase, a string matching algorithm is employed to perform pattern matching. Here Aho-Corasick algorithm is applied to all substrings drawn from the cutout part. This way of applying string matching is ten times faster than the scalar string matching using Aho-Corasick algorithm.

ii. *Sidhu's Algorithm for String Matching using Hardware Technology*

The algorithm is grounded on non-deterministic finite state machine (NFSM) for regular expression matching. In the field of computing, regular expression gives a concise meaning to "match" [21]. The pattern can match one or more text strings. A non-deterministic finite state machine or automaton is a state machine resembles a directed graph that exhibits different states represented by nodes and edges designate character or empty string. The algorithm works by generating regular expressions for every string and NFSM examines the input at a speed of one byte at a time. This approach needs a time of O(m), m symbolize pattern length. NFSMs are tough to implement and requires rebuilding every time a string is added making it complicated.

*c) String Matching Based on FM-Index*

Despite of many algorithms presented on string matching, the present attempt to solve string matching problem uses FM-index technique that concatenates the attributes of suffix array and Burrows-Wheeler transformation [22]. To understand the working of this architecture the above concepts has to be acknowledged. The next segment of this section presents a detailed discussion of it.

i. *2D-LARPBS*

It represents two-dimensional LARPBS. The model has the system's buses arranged in a two-dimensional set up that makes communication among buses more effective [23]. It can be use for the design of both exact and approximate string matching. The construction of these algorithms is based on Hamming distance.

ii. *Hamming Distance*

Hamming distance measures the amount of inequality between two strings. It can be applied for error detection and correction. For any two strings it gives the number of the corresponding characters that are dissimilar. Using this measure the knowledge of closeness of two strings can be known and thus gives an idea about the operations to be done to obtain one string from another.

a. *Formal Definition of Hamming Distance*

For strings A and B with same length k, the Hamming distance H (A, B) is given by

H (A, B) = no. of positions where A[i]!=B[i] and $0 < i < k$.

## III. Text Processing Techniques

Making text ready to be scanned for string search so that it helps yield reduced search time is text processing. In the previous chapter's literature it was determined that to improve time complexities of string

matching approaches parallelization has to be adopted. The root lead towards this starts with divide and conquers procedure and dynamic partition techniques intended for parallel processing, and are presented in the current chapter.

*a) Divide and Conquer Paradigm*

i. *Introduction*

Decomposing a complex problem into two or more smaller sub-problems until a simple portion is obtained for easy solvability is dividing and conquer paradigm as the name suggests [2]. If the problem is easy it can be solved directly but if a complicated problem persists then breaking into its small instances and solving each instance independently resembles divide and conquer strategy. The solutions to the smaller versions of main problem are clubbed up to attain actual solution to the original instance. It adopts the recursive division while undergoing the breaking up of a problem. The goodness of using divide and conquer strategy lies in the point that it stands as a powerful tool in solving conceptually tough problems [12]. The algorithms implementing divide and conquer paradigm are often found to be efficient. It is also considered an apt algorithm to be executed in multi-processing environment as distinct sub-parts of a problem can be executed in parallel on different machines. The general stages of this strategy would be divide, conquer (solve) and unite.

*b) Generic Divide and Conquer Algorithm*

*Input:* Problem P and n = Size (P)

*Output:* S = Solution (P)

Begin

Step 1: If n is small Solve (P)

Step 2: Else divide P into sub-problems p1 and p2 of lengths n1 and n2 respectively such that

$$n1 \approx n2 \approx n/2$$

Step 3: Conquer

$$S1 \leftarrow \text{solve } (p1, n1)$$
$$S2 \leftarrow \text{solve } (p2, n2)$$

Step 4: Unite solutions to obtain actual solution

$$S \leftarrow \text{unite } (s1, s2)$$

End

## IV. Hypercube Network Topology

A hypercube is a geometrical figure in four or more dimensions similar to a cube in three dimensions with all its edges having equidistant from their respective nodes. For an n-dimensional hypercube, there are $2^n$ vertices and $n*2^{n-1}$ number of edges. In a network, a node is a connection point or a redistribution point, more formally in a physical network it is an electronic device that can send, receive or forward information over the communication channel to other nodes in the network where as edges provides access to the network and also involves in transmitting information in a network over the nodes [24]. A router is an example of an edge device in a network.

In computer science, a hypercube network is a configuration of multiple parallel processors having distributed memory such that the locations of the processors are analogous to the vertices of a mathematical hypercube and the links correspond to the edges. For an n-dimensional hypercube, as mentioned above, it has $2^n$ processing nodes and $n*2^{n-1}$ edges coupled in an n-dimensional cube network. The $2^n$ nodes are designated by binary numbers from 0 to $2^n$-1. The nodes are connected by links responsible for intercommunication. The two nodes are connected if the binary numbers assigned to it stand apart by exactly one bit position.

*a) Message Transmission in Hypercube Network*

Every node in the network has the ability to send, receive and transmit data to other nodes. The information that is passed is in the form of packets. Every node is represented by a unique id that is presented in the binary form. For n-dimensional hypercube, every node is represented in n-bits.

*b) Algorithm to find the shortest path for transmitting message from source node to target node*

*Input:* Source node $S_{n-1}S_{n-2}\ldots\ldots S_3S_2S_1S_0$, Target Node $T_{n-1}T_{n-2}\ldots\ldots T_3T_2T_1T_0$.

*Output:* Path of transmission.

Begin

Step 1: Equate each bits of the source node with target node beginning with the right-most bit that is from the LSB of Source node. Formally, compare $S_0$ with $T_0$.

Step 2: On encountering inequality, if S0!=T0 then complement the respective bit to get the next intermediate node for transmission. The next bit would be $S_{n-1}S_{n-2}\ldots\ldots S_3S_2S_1S_0{}^c$.

Step 3: Repeat steps 1 and 2 for each of all the bits preceding LSB exclusively till the target node is obtained.

End.

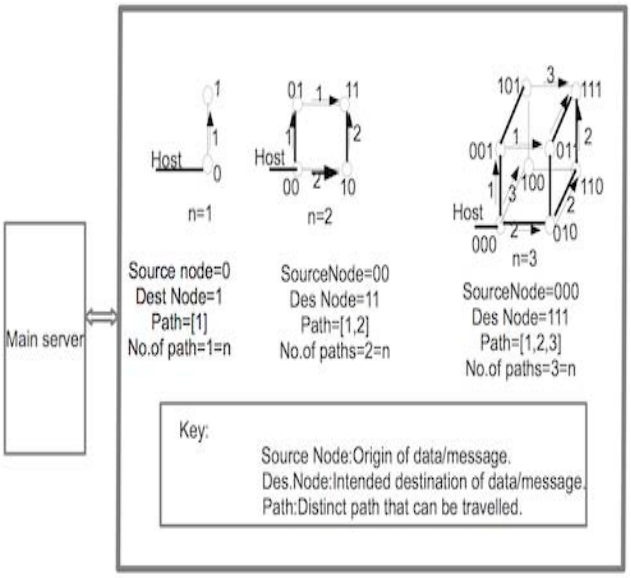



*Figure 1 :* Distinct Communication Patts.

In the above figure circles shows the node and arrow mark shows the path which connects the server/nodes. The main server which is connected from the outside the hypercube network servers, such that connection establishes from host node to destination node and path is established to broadcast the text files and pattern files with the help of hypercube program. Path direction along with the node numbers are shown in the above diagram for n=1 means two node H.C, n=2 means four node H.C and n= 3 means eight node H.C.

## V. Experimental Setup

Experimental setup required for the above implementation is more processors P(at least four) connected with hypercube model on INTERNET of either similar systems (homogeneous ) or dissimilar systems (heterogeneous). P processors where 0<P<5 and time, by taking K patterns where 0<K<4 as key factor, before conducting test [2].

*a) Parallel kumar viswanadha and Boyer Moore String Matching Employing Overlapped Text Partitions*

The algorithm implements Boyer Moore string matching algorithm in a parallel environment. The input text string is sliced into *'i'* subtexts such that each text partition holds (n/P)+m-1 text string characters with m-1 text characters overlapping in each partition, here P refers to the number of processors in the topology, m and n being the lengths of text and pattern string respectively [12] .The number of sub texts obtained after partitioning the text string using the above formula equals the number of processors allotted in the architecture, i.e., *i=P,* thereby representing the static allocation of the processors. The complete idea behind the working of this procedure can be well understood by the algorithm given below.

i. *Parallel kumar viswandha Boyer Moore Algorithm*

Begin

Step 1: Input on the user interface text file of size n, pattern file of size m and number of processors (P) available.

Step 2: Undergo text file division into ' *i* ' number of subtexts, each i contains (n/P)+m-1 text characters using m-1 overlapping text characters. The divided sub text files are stored in a directory.

Step 3: Broadcast these sub text files to each processor in the topology.

Step 4: Each Processor searches the pattern string in the given Sub text file using the Boyer Moore Algorithm and sends back the result.

Step 5: *Boyer Moore Algorithm*

Begin

Step 5.1: A window of size pattern slides over the text Scanning m elements of text string with the pattern string of length m from right to left.

Step 5.2: On a mismatch use the longest shift distance of the bad character heuristic and Good suffix heuristic. Window shifts are carried till n-m+1$^{th}$ position is attained on the text string.

Step 5.2.1: The bad Character heuristic states that the mismatching text character termed as bad character of the corresponding pattern character is searched for the rightmost occurrence in the left portion of the pattern window, if found the bad character is aligned with it. If occurs nowhere in the pattern then m characters of the text can be skipped.

Step 5.2.2: According to the good suffix heuristic, the suffix appending the bad character is searched in the left portion of the pattern and thus aligned if found else skip m characters of the text string.

Step 5.3: On a successful match of all the pattern characters with the text characters in the window, locate the pattern string and continue matching for the next occurrence skipping m characters of the text.

Step 5.4: Repeat steps 5.1 to 5.3 till n-m+1 position of the text string.

End

Step 6: Each processor stores the sub results and sends back to the main program to sum up the obtained results.

End.

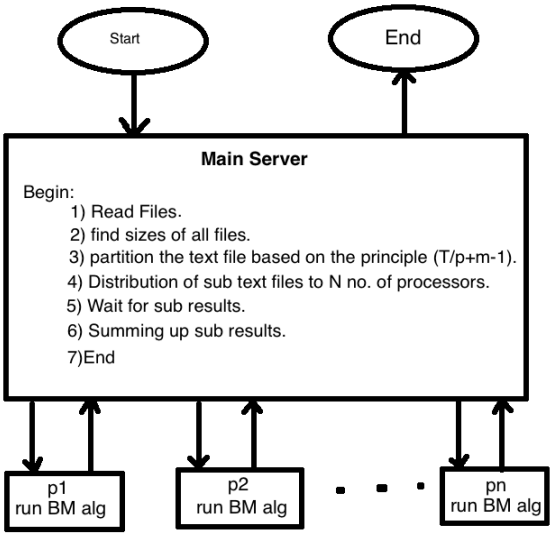



*Figure 2 :* Overlapped Partition Algorithm on Distributed Network

## VI. Result Analysis and Discussions

We have considering three files for the implementation discussed in the previous chapter such as $f_1$ of size 1 Mb, $f_2$ of size 2 Mb, and $f_3$ of size 3 Mb from TREC- 05psn datasets and TREC-09ps micro biology datasets. The pattern files are $p_1$, $p_2$, $p_3$ with respect to those three files [2, 10, 12]. Here bytes mean number of characters. Time is measured in milli seconds $p_{i,j,}$ represents pattern i in file j.

Example : $p_{1,1}$ gives pattern 1 in file 1 ($f_1$),
$p_{1,2}$ gives pattern 1 in file 2 ($f_2$).

File 1 The pattern files that are searched in the text file $f_1$ are $p_{1,1}$ of size 3 bytes, $p_{2,1}$ of size 10 bytes, and $p_{3,1}$ of size 15 bytes has to be found using KV-Boyer moore Exact string matching algorithlm and as well as KMP Algorithms.

File 2 The pattern files that are searched in the text file $f_2$ are $p_{1,2}$ of size 8 bytes, $p_{2,2}$ of size 4 bytes, and $p_{3,2}$ of size 20 bytes has to be found usingKV- Boyer moore Exact string matching algorithm and as well as KMP Algorithms.

File 3 The pattern files that are searched in the text file $f_3$ are $p_{1,3}$ of size 3 bytes, $p_{2,3}$ of size 7 bytes, and $p_{3,3}$ of size 19 bytes has to be found using KV-Boyer moore Exact string matching algorithm. The test is conducted for three text files against three patterns of different sizes for Both the Boyer Moore and KMP and The results are shown in tables. The results of KMP algorithm are beyond the scope of this paper.

Results of KV-BM string Matching Algorithms are basically taken from output file and from instant graphs. The program was designed and implemented such that, it generates the instant graph (Bar chart) based on the No. of processors which are represented on X-axis and time on Y-axis in milliseconds along with no. of occurrences of pattern against each processors shown in the figures 3, 4, and 5. The Bar chart Graph uses the multi-colors to separate the each processors from the other, on the top of the bar No. of occurrences are mentioned with numeric number in braces. The program gives the output results in the form of text file along with the instant graphs. The output results text file gives the test parameters like start time ,end time and elapse time , along with the time taken for reading the text file and broad costing timings of sub text files . It also gives other kinds of output parameters called as position of the pattern occurrences and size. The figure.7 in tables we shown only the elapse time and average time of the processors involved in milliseconds, along with the no. of times the pattern is occurred . 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 table shows that for each pattern, as the No. of processors increases the time reduces and accuracy Increases. The graph's 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.

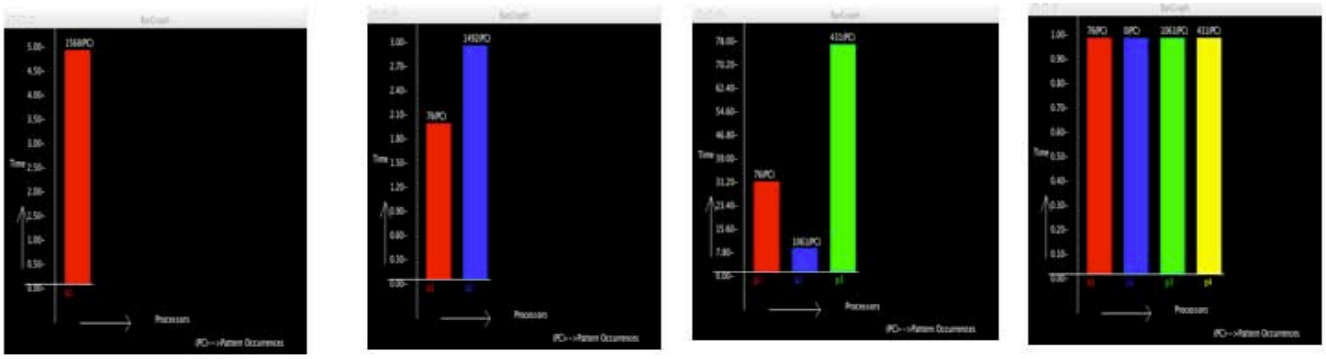


Figure.(3) :Instant graph for Pattern.1(gov) on (a) Single processor ,(b) Two Processors(c) Three processors (d) four processors

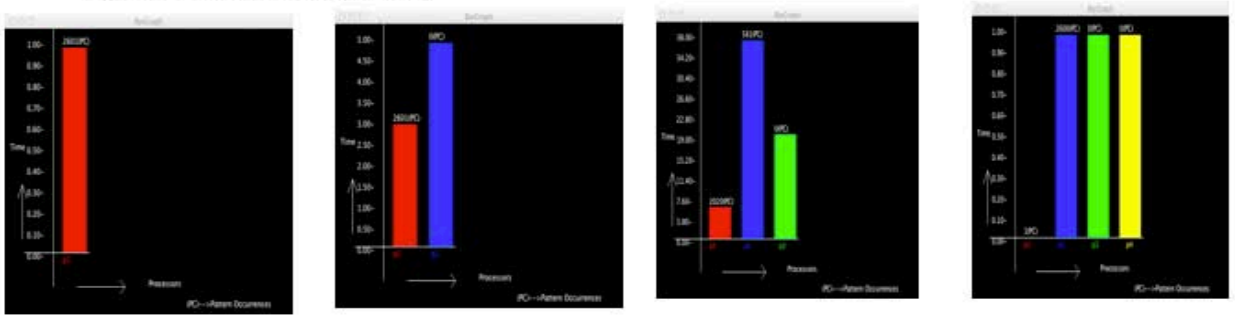


Figure.(4) :Instant graph for Pattern.2(graylabs) on (a) Single processor ,(b) Two Processors(c) Three processors (d) four processors

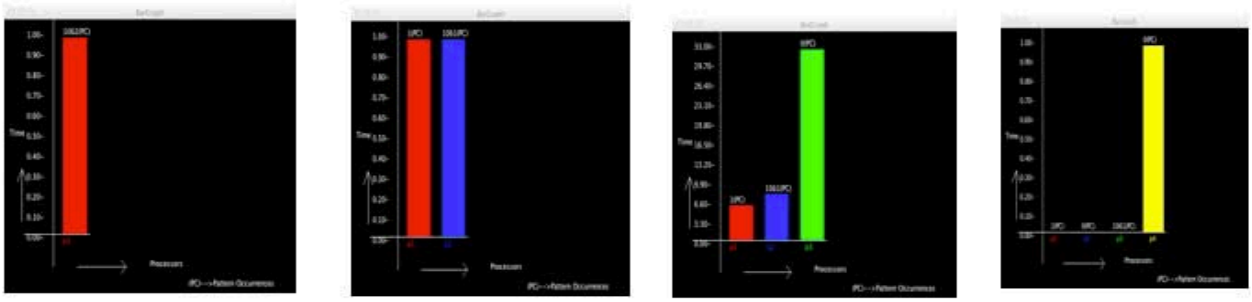


Figure.(5) :Instant graph for Pattern.3 (sis_gov_org)on (a) Single processor ,(b) Two Processors(c) Three Processors (d) four processor

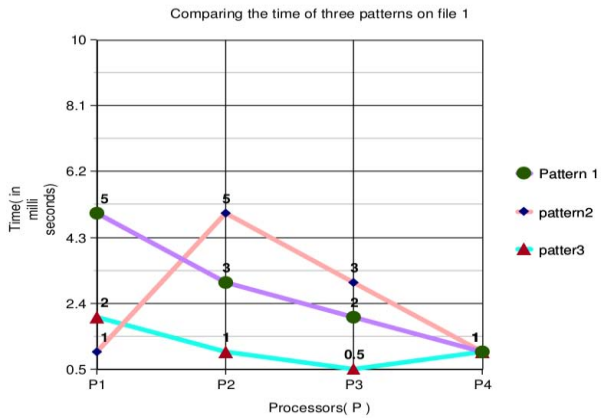
Table1: BM-SM Shows the variation of time among the processors for File 1

| S.No | Pattern with size | Single processor | | Two processors | | | Three Processors | | | | Four Processors | | | | | No.of times pattern occurred |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | Elapse Time of P1 | Average time | Elapse Time of P1 | Elapse Time of P2 | Average time | Elapse Time of P1 | Elapse Time of P2 | Elapse Time of P3 | Average time | Elapse Time of P1 | Elapse Time of p2 | Elapse Time of P3 | Elapse Time of P4 | Average time | |
| 1 | P1-(3bytes) | 5 | 5 | 2.1 | 3 | 3 | 2 | 2 | 1 | 2 | 1 | 1 | 1 | 1 | 1 | 1568 |
| 2 | P2-(7bytes) | 10 | 10 | 3 | 5 | 5 | 2 | 3 | 2 | 3 | 1 | 1 | 1 | 1 | 1 | 2601 |
| 3 | P3-(10bytes) | 1 | 1 | 1 | 1 | 1 | 0.5 | 0.5 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 1026 |

Figure.(6) :Results are tabulated for elapse time along with number of occurrences

*Figure 6 :* KV-BM algorithms results are tabulated for four processor against three patrern files

*Figure 7 :* KV-BM algorithms results on line graph for comparison of three pattern files




This graph is constructed online by feeding the results from the above table. It is evident that the pattern 1 is of size 3 bytes and text file of size 1MB takes the 4.8 ms time to search the pattern but as the No. of processors increases it reduces to 0.5ms. In case of pattern 2 the size is 7bytes and it is also behaves similar to pattern 1 in case of more No. of processors but, for pattern 3 of size 10 bytes search time reduces drastically as the pattern size increases and as well as No. of processor increases .Hence our experimental results give excellent out puts and we also conducted more experiments but , results are not presented due space problem Theoretically discussed.

## VII. Conclusion

In this paper we have 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. In near future we also produce new results by conducting more no. of experiments using the similar setups.

## References Références Referencias


1. Roi Blanco, B.Barla cambazoglu "8th LSDS-IR'10 – ACMSIGIR Form, Vol 44 No. 2 Dec-2010.
2. S. Viswanadha Raju, K.S.M.V. kumar, "Implementation of String matching on Multiprocessors using divide and Conquer Technique" in 2011 3rd international Conference on Machine Learning and Computing (ICMLC) WWW.Elsevier.com, Dec 2011.
3. D. E. Knuth, J. Morris, and V. Pratt, "Fast Patern Matching in Strings," SIAM J. of Comput., Vol. 6, pp. 323-350, 1977.
4. http://www.worldwidewebsize.com
5. www.lsdsir.org-page-id=21
6. R.S. Boyer and J.S. Moore. A fast String Searching Algorithm Communications of the ACM, 20(10):762–772, 1977.
7. Awsan Abdulrahman Hasan and NurainiAbdul Rashid "Hash-Boyer Moore-Harspool string Matching Algorithm for Intrusion Detection Systems" INCNCS-2012, IPCSIT vol. 35 (2012), Singpore.
8. V. Aho and M. J. Corasick. Efficient String Matching: An Aid to Bibliographic Search. Communication of the ACM, 18(6): 333–340, 1975.
9. M Allen, B. Wilkinson, "Parallel Programming: Techniques and Applications using Networked Workstations and Parallel Computers", Prentice Hall, 1999.
10. S. Viswanadha Raju and A. Vinaya Babu, 2006, "Optimal Parallel Algorithm for String Matching on Mesh Network Structure", International Journal Applied Mathematical Sciences, 3 No.2, 167-175.
11. Snort. www.snort.org.
12. S. Viswanadha Raju, S.R. Mantena, A. Vinayababu and GVS Raju, 2006, "Efficient Parallel String Matching Using Partition Method", Proc PDCAT-2006, IEEE Computer Society, 281-284.
13. Herbert Scheldt's (Osborne) Java 2-Complete Reference, 5th edition-2008.
14. RMI:http://java.sun.com/products/jdk/rmi/, http://www.ece.vill.edu/~khenry/rmiapp/
15. Bi Kun, Gu Nai-jie, Tu Kun, Liu Xiao-Hu and Liu Gang "A Practical Distributed String matching Algorithm Architecture and Implementation WASET vol.19, pp.156-162, oct. 2005.
16. Zvi Galil, "A Constant-Time Optimal Parallel String-Matching Algorithm," Journal of the ACM, Vol. 42, pp. 908-918, July 1995.
17. Hiroki Arimura, Atsushi waraki Ryoichi Fujino and Stno Arikawa "A Fast Algorithm for Discovering optimal String Parrens in Large Text Databases".
18. Jin Hwan Park and K.M. George, "Parallel String Matching Algorithms Basedon Dataflow", Computer Science Department, Oklahoma State University, Stillwater, USA.
19. Cooks and Rivest "Two-way pushdown automata for string matching" journal of IEEE.1977.
20. Y. Mishina and K. Kojima string-matching algorithms for vector processing and its implementation in proceedings of 1993 IEEE international conference on computer design (ICCD'93) 1993.
21. R. Sidhu's V.K Prasarna "Fast regular Expression Matching using FPGA's" In IEEE symposium on FPCCM; Rohnert park, CA USA, April 2001.
22. Edward Frenandez, W Najjar and S Lonardi "String-matching in Hardware using the FM-Index". In IEEE 1998.