An Optimal Algorithm for Matching String Patterns in Large Text Databases

K.s.m.v.Kumar†, S.Viswanadha Raju††, and KA.Govardha

†Assoc.Professor in Department of CSE ,SIET ,IBP, AP-501506, India
††Professor in Deparment of CSE, JNTUH University, Jagityal,AP-505501, India
†††Professor in Department of CSE, JNTUH, Hyderabad, Ap-500085, India

Summary
Modern parallel distributed string matching algorithms are always based on networked computation model. Those algorithms depends on the cost optimal design and the theoretical speed. The motive of current research challenge**s** and identified the new directions I.e distributed environment where in which the given text file is divided into subparts and distributed to P1 to PN no. of processors organized in parallel environment called as hypercube network. Based on the distributed memory machine string matching algorithms proposed by CHEN and BI-Kun, a optimal algorithm for matching string patterns in large text databases over parallel distributed hypercube networked architecture is proposed in this paper. And also an improved parallel string matching algorithm based on a variant Boyer-Moore algorithm is presented. We implement our hype and integrated algorithms on the above architecture and the experiments are proven that it is truly practical and efficient on parallel distributed Hypercube networked model. Its computation complexity is O(T/p + m-1), where T is text file of length n characters, and m is the length of the pattern, and p is the number of the processors.

Keywords:
Boyer-Moore algorithm, distributed networked model, parallel string matching, optimal design, and patterns.

1. Introduction

Pattern matching diversely used in many Applications of computer sciences and it received much attention over the years due to its importance in various applications such as text processing, Search patterns , information retrieval, computational biology, address lookup, and intrusion detection [3]. All those applications require highly efficient and fast-search algorithm to find all the occurrences of a given pattern in the text. Rapid growth of abundant information makes necessary to have efficient methods for information retrieval. Now a days commercial search engines are totally dynamic and their web indexing is done on a few data centers [6]. It is necessary to come up with scalable indexing, searching and query processing techniques for next generation IRS in the nearest future. The web comprises wide variety of content in the form of unstructured meta data, databases, Google maps, images, Audio/videos and textual documents etc.[3]. The main challenge of present day IRS design is scalability. A recent studies says that the number of servers required by a search engines to keep up with the load in 2013 may be in the order of millions as such the text size is increasing exponentially, tens of billions of pages[3,6]. Hence it is very urgent to develop a truly distributed very large scale systems that enables fast and accurate search over very huge amount of information content [12].

Actually the string Matching problem is classified into two major categories, known as Exact and Approximate string matching. The Exact string-matching was Further sub- divided into single String Matching and parallel String Matching. Similarly approximate string matching is divided into K-mismatches problem and K-differences problem [20]. All these problems can be solved by either software-based solutions or Hardware- based solutions. Since software-based solutions are slower and less efficient, hardware-based solutions are highly preferred. This section continues with a brief discussion and classification of all early string matching algorithms was covered.

The Knuth-Morris-Pratt(KMP) algorithm[2] and the Boyer-Moore(BM) algorithm[7] both are more familiar single string matching algorithms. The KMP algorithm gives guaranty that it is independent of text size and linear worst-case execution time in the pattern length, and the worst-case computation complexity is $O(n + m)$; on the other hand, the BM algorithm provides near optimal average case and best-case behavior, and it only needs $O(n / m)$ comparisons in the best case. More than hundreds of algorithms has been proposed based on the KMP or BM algorithm, such as BMH algorithm [24], QS algorithm Rabin-krap, fast search, Bi-kun [20,8] and so on [12]. Based on the critical observation on the characteristics of the “bad character” and “good suffix” in BM algorithm, Cantone and Faro [25] designed a more efficient algorithm called Fast Search. Though it keeps the good characteristics of the BM algorithm, its worst-case computation complexity is still $O(n \times m)$. BI-kun proposed

Manuscript received June 5, 2013
Manuscript revised June 20, 2013

an improved single string matching algorithm by making some modifications on the algorithm proposed by Cantone and Faro[25]With those Modifications, the good characteristics of the algorithm are still preserved, while the worst-case computation complexity has reduced from $O(n \times m)$ to $O(n + m)$

In this paper we mainly focus on string matching on distributed environment called as hypercube network model using RMI method [10]. Given a pattern length may be smaller or bigger, we wish to count how many times it occurs in the text and its positions of occurances. For pattern matching we used two distributed algorithms called as integrated algorithm (KMP and Boyer-Moore) and hype algorithm (Boyer Moore string matching on hypercube network). The text file is processed in two ways, one is non-overlapping and second is overlapping text partitioned processing[8]. In both the cases integrated and hype algorithms are applied for string matching and the remote server will be invoked using JAVA RMI method on hypercube networked model to reduce the search time[1,8,10] . Our proposed Algorithm result performances and time complexities are compared with others Algorithms called as BMH, Fast-search, and BI-Kun Algorithms. The performance differences are clearly discussed in the result analysis chapter [20,29] .The proposed algorithm computation complexity is O(T/p + m-1), where T is text file of length n characters, and m is the length of the pattern, and p is the number of the processors. It shows that the search time is inversely proportional to No. of processors.

The paper is organized as follows Chapter II deals with Related work of single and parallel string matching algorithms in both software and hardware approaches, Chapter III deals with text processing techniques. Chapter IV explains about the Distributed Hypercube Network Architecture And Algorithms. Chapter V presents Experimental Setup. Result analysis and discussions were discussed in Chapter VI and ChapterVII is conclusion. Acknowledgements and the References are added at the end.

2. Related Work

In This chapter we discussed the all the algorithms which gives an overview that, how often the algorithms used in achieving the desired information along with its time complexities. String matching can be achieved by designing algorithms in two categories namely, exact string matching algorithms that locates exact match of the pattern in the text string. and approximate string matching algorithms that finds closest possible match of pattern in the text with some mismatches. We can address Exact string matching problem in two ways single string Matching and parallel string matching, can be implemented using software based approach or hardware based approach[7]. Software based algorithms are slow in performance compare with hardware based algorithms [16,22]. Hardware based solutions to string matching provides efficient data storage and fast matching .

2.1 Software-based Single String Matching Algorithms :

In 1972, Cook experimented string matching problem 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[5]. In Succeeding with Cooks experiments in 1977 Rivest determined that every string matching algorithm must go through at least n-m+1 comparisons at worst case. This shows there is no solution of obtaining a sub linear n worst time in solving the issue. 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[2]. It is the first linear pattern matching algorithm discovered with a run time of O(m+n).

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[9].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 weakness of this approach as the text has to be read for k times.

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 familiar algorithms and also stands as a benchmark for string matching process[7]. The algorithm compares pattern within a sliding window over a text string, applying right to left scan of alphabets inside the window where as the window slides from left to right over the text. The goal of this algorithm is to skip certain fragments of text that are not good for comparison. This decision is taken by placing the window in left alignment with text. The algorithm begins to 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.

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[29]. 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. Horspool explained this problem in two cases.

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.

With KMP and the BM string matching algorithms, lots many algorithms are developed to improve the efficiency of the original algorithms. When the text size is large enough, Horspool[24] suggested using only "bad character" rule shifts when a mismatch occurs. This algorithm is practically faster when the text size is big because it does not need to make a comparison between the "bad character" shift and the "good suffix" shift which is used to shift the pattern when a mismatch occurs. Hundreds of other algorithm variants based on KMP or BM and a fairly complete bibliography are available on the web site [27]. Cantone and Faro [25] discovered that "the Horspool bad character rule leads to larger shift increments than the good suffix rule if and only if a mismatch occurs immediately, while comparing the pattern p with the window", so they suggested using the "bad character" rule when the mismatch occurred immediately in comparing the text, otherwise, using the "good suffix" rule. If the pattern is periodical, the worst-case computation complexity of BM algorithm is $O(n \times m)$, so are the BMH algorithm, the QS algorithm and the Fast-Search algorithm[29]. Galil [21] proposed an algorithm to improve the worst case running time of the BM algorithm, and he proved the improving BM algorithm is O(n+m) in worst case.

2.2 Software Based Parallel String Matching Algorithms:

The first optimal parallel string matching algorithm was proposed by Galil [21]. On SIMD-CRCW model, this algorithmis required $n / \log n$ processors, and the time complexity is $O(\log n)$; on SIMD-CREW model, it required n/log ^{2}n processors and the time complexities is O (log ^{2}n).Vishkin [28] improved this algorithm to ensure it is still optimal when the alphabet size is not fixed. In [20], an algorithm used $O(n \times m)$processors was presented, and the computation time is $O(\log \log n)$. A parallel KMP string matching algorithm on distributed memory machine was proposed by CHEN[26]. The algorithm is efficient and scalable in the distributed memory environment. Its computation complexity is $O(n / p + m)$, and p is the number of the processors . Cantone and Faro [25] designed a more efficient algorithm called Fast Search. Though it keeps the good characteristics of the BM algorithm, its worst-case computation complexity is still $O(n \times m)$. BI-kun proposed an improved single string matching algorithm by making some modifications on the algorithm proposed by Cantone and Faro [25]. With those modifications, the good characteristics of the algorithm are still preserved, while the worst-case comp-utation complexity has reduced from $O(n \times m)$ to $O(n + m)$.

2.3. Hard ware Based Single String Matching Algorithms:

Mishina Algorithm

Mishina et al 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[14]. 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.

Sidhu's et al proposed a Algorithm for String Matching using Hardware Technology. The algorithm is grounded on non-deterministic finite state machine (NFSM) for regular expression matching[15]. A regular expression is a pattern that matches one or more strings of characters. This approach needs a time of O(m), m symbolize pattern length.

Tuck et al. projected few alterations to Aho-Corasick algorithm to lessen the memory needed to storage of malicious strings and also made worst case time better. This compressed data storage is projected to accommodate the data in the cache of commodity processors or on-chip SRAM, abbreviates Static Random Access Memory[13]. Experiments showed that the compression techniques specified above could run in 50 times less amount of database size over by Aho-Corasick algorithm.

2.4 Hardware Based Parallel String-Matching Algorithms:

String Matching based on FM-Index

FM-Index is a full text indexing procedure that combines Burrows-Wheeler Transformation and suffix array to

locate pattern. This technique was given by Paolo Ferragina and Giovanni Manzini in 2000.The benefit of using this approach is FM-Indexing is very space efficient and extremely fast. FM-Index has two pointers namely *T* and *B* that initially points at top and end of the suffix array[16]. T and B pointers point to the indices that refer suffix locations where a pattern can occur. The pointer 'T' points to a suffix location where there is a possibility of first occurrence of pattern and 'B' points to a suffix location where a pattern can occur for the last time. If the index pointed by T is greater than or equal to an index pointed by B then the pattern does not occur in the string.
In 1979, Commentz- Walter brought up a format by concatenating Boyer-Moore algorithm with the above method so that an automaton for opposite set of keywords is build. This worked well for small length pattern string that original Aho-Corasick algorithm, running in a quadratic amount of time in worst case[9]. By adopting preprocessing, it achieved a linear time of O(n). An identical style of addressing pattern matching was given by Kim and Shawe Taylor in 1992 employing Boyer Moore- Horspool [7,24]algorithm by Baeza-Yates, 1990.Multiple-String search using shift-add algorithm by Baeza-Yates and Gonnet, in 1989, 1992 presents a flexible numerical access taking O ([n/w]*m) time, w representing computer word size in bits. If m < w, then text is scanned in a linear time.For searching a text repeatedly to locate occurrences of heterogeneous pattern it is good to construct an auxiliary text index. One of this is presented by Weiner, 1973 or a suffix tree that can be put up in a linear time .

3. Text Processing Techniquices

3.1. Overlapping Text Partition

Making text ready to be scanned for string search so that it helps yield reduced search time is text processing. The root lead towards this starts with divide and conquer procedure and dynamic partition techniques intended for parallel processing. For a text of length *n* and pattern of length *m*, and $n \geq 2m$ cut text into $n-m+1$ partitions. The length of each partition equals the length of the pattern, *m.* To make text partitions overlap, the next successive partition starts from the position $| B_i |-m+1$ of the current partition/block on the text where B_i is the length of the current partition[8]. Assign each partition and the pattern string to one of the $n-m+1$ processors. Each processor looks for the equality of the partitioned text substring and the pattern using any one of the linear string matching algorithms and returns *1* if a match is found and *0* on mismatch.

3.2. Non-Overlapping Text Partition

Divide and conquer paradigm using non-overlapping partitions is the other way of solving string matching. Non-overlapping text partition results in a drawback of pattern bifurcation among partitioned sub texts; this loses the pattern string continuity. For example for a text string ABCDCDAA, and pattern string CDCD, the non-overlapping text partitions of pattern length would be ABCD, CDAA. The Pattern 'CDCD's search in these substrings would result in non-occurrence of pattern string but pattern very much exists[8,10]. To overcome the above problem, instead of slicing text, decompose pattern string into its partitions employing divide and conquer strategy and addressing string matching in multi processor environment.

4. Distribured Hypercube Network Architecture and Algorithms

Experimental setup required for the above implementation is more processors P(at least four) connected with hypercube model on INTERNET of either similar systems or dissimilar systems[1,8,10]. P processors where $0<P<5$ and time, by taking K patterns where $0<K<4$ as key factor, before conducting test.
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[8]. The two nodes are connected if the binary numbers assigned to it stand apart by exactly one bit position.

4.1 Message Transmission in Hypercube Network:

Parallel broadcasting n-dimentional network assumption: For degree d forwarding, we assume inputs are arranged in d different input streams as we described in algorithms 2 and 3 [8,10,12].

Let subtext i represents the ith (from left) input stream of size (T/p+m-1) and let m be the size of input pattern and d be the degree of parallelizium.

```
Procedure Hypercube(myid,input text,input pattern, logP,
                                                output)
begin
      Node(state)= input                    //local node.
          For   i= 0 to logP-1                  // repeat log
```

P times
dest= myid XoR 2^i // determine common server
send state to dest // Exachnge data
receive message from dest
node /state =OP(state, message)
end for
Output=state(input node)
end.

Algorithm 1: In the hypercube communication, the above procedure is executed by each task in a hypercube communication structure, with logP denoting the size of the hypercube ($P = 2^{logp}$) and myid is the tasks identified in the range $0....2^{logP-1}$. XOR denotes an exclusive or operation and OP is the user specified operator used to combine local data with data received from the i^{th} neighbor in the hypercube .

4.2. String-Matching Algorithms:

The integrated and hype algorithms 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
and non-overlapping fashion 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[8].

Hype Algorithm:(Overlapped String Matching on hypercube network)
Begin
Step 1: Inputs :Text file T of size n, pattern p file of size m and No.of processors (P).
Step2**:** Text file division into ' *i* ' No.of subtexts, of size (n/P)+m-1 text characters using m-1 overlapping text characters are stored in a directory.
Step 3: Broadcast these sub text files to each processor in the hyper cube network 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*:* 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 6 *:* 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 7 *:* Repeat steps 5.1 to 5.3 till n-m+1 position of the text string.
Step 8: Each processor stores the sub results and sends back to the main program to sum up the obtained results.
End.
Integrated Algorithm:(Non-Overlapping String-Matching)
\
Algorithm (pattern, text)
Begin
Step1:Divide text string into equal length partitions. For odd length text, fill the empty space with null characters.
Step2:Broadcast subtext and the pattern string to the processors in the network.
Step3:Parallel pattern search, Each of the processor attempts to locate the pattern in the assigned subtext using Boyer Moore Algorithm.
Step4:Form the connector strings Join the first and last m-1 characters of the adjacent partitions giving a new connector string.
Step4:Sequential search Applying Boyer Moore algorithm on the above strings locates the existence of the pattern.
Step5:Sum Up The results generating from multi processing search and the single search is summed up to know the total number of occurrences of the pattern.
End.

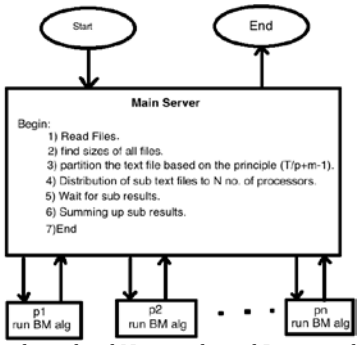

Figure.1 Overlapped and Non overlapped Partition algorithm on distributed network.

5. Time Complexity Analysis

The time complexity of our improved Parallel single string matching algorithm is $O(n / p + m)$, because the text of length n is partitioned and then assigned to each processor before processing by each processor, the length of the assigned text is n / p , and p is the number of the processors, at most $2m - 2$ characters need to be checked further, so the complexity is $O(m)$. So the total time complexity of the [25] algorithm is $O(n / p + m)$. Practical distributed string-matching algorithm architecture proposed by Bi-kun, the pattern of length m need to be broadcasted to all of the processors[29]. If Binary-tree communications strategy is used, the

communication compleity is O(m logP) and total time complxity is estimated as O(m logP+m).
Hype and integrated algorithms on the distributed hypercube networked architecture are implemented and the experiments results are proven that it is truly practical and efficient. Its computation complexity is O(T/p+m-1), where T is text file of length n characters , and m is the length of the pattern, and p is the number of the processors.

6. Experimental Results and discussions

We have considering one text files for the implementation discussed in the previous chapter such as f_1 of size 3 Mb from TREC- 05psn datasets . The pattern files are p_1, p_2, p_3 with respect to files(23, 5,6 bytes). Here bytes mean number of characters. Time is measured in milli seconds

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

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

File 1 The pattern files that are searched in the text file f_1 are $p_{1,1}$ of size23 bytes, $p_{2,1}$ of size 5 bytes, and $p_{3,1}$ of size 6 bytes has to be found using Hype Exact string matching algorithlm and Integrated Algorithms.

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(communication) timings of sub text files . It also gives other kinds of out put parameters called as position of the pattern occurrences and size. The figure.2 and 4 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 .

6.1 Hype Algorithm Results:

Table 1: Hype Algorithm's Results are Tabulated.

S.No	Pattern with size	No of times pattern occurred	Single processor Computing time		Two processors Computing time			Three Processors Computing time				Four Processors Computing time				
			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-(23bytes)	276	13	13	8	1	8	4	0	1	4	5	0	1	1	5
2	P2-(5bytes)	985	94	94	41	22	41	13	3	1	13	8	1	1	1	8
3	P3-(6bytes)	2347	21	21	10	10	10	4	1	1	4	1.6	2	1.4	1	2
Communication Time			0 milli seconds		4.24 milli seconds			6.43 milli seconds				9.35 milli seconds				

Figure 2: Hype Algorithm's variation of time among the processors for File 1

Table 1 shows the output results of file1 of 2 MB, for three patterns of different sizes P1 (23 bytes), P2 (5 bytes) and P3 (6 bytes). The table gives the understanding that the algorithm takes less time when number of processors increase and the time also varies with the pattern size. It is evident that, If pattern size increases and also No.of occurrences increases then the computing time reduces for given text file. The table gives the reading that the size of the pattern is proportional to the elapse time. In general it can be concluded that the results are effective when number of processors increases. The table also shows the communication time and pattern sizes .
The variation in timings for three search patterns are compared with the help of combined graph constructed using on line graph method shown above. In the above graph with the processors on X-axis and time on Y-axis in milliseconds, shows the three patterns in three different lines with different colors and timing mentioned with numerics on graphs .As the number of processors increases each of the pattern's search time reduced to a better extent. The pattern of bigger sizes(in case of pattern 1) the computing time is very less shown in the above graph. This algorithm works well for any pattern size with respect to No. of processors involved and the performance will be improved further for bigger pattern sizes and alsoif more no. of processors involved.

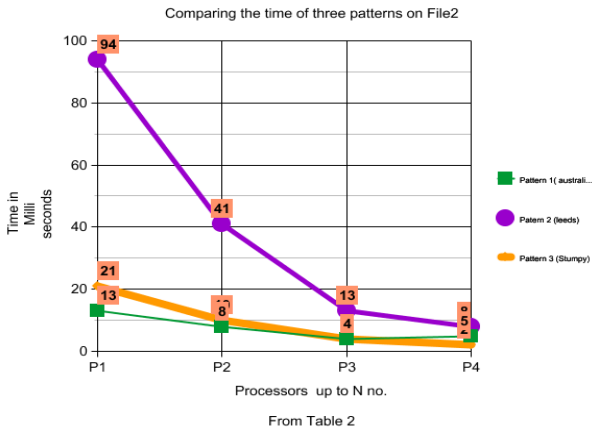

Figure 3 :On line graph for Comparison of three patterns for text file 1

6.2 Integrated Algorithms Results:

Table 2: Integrated Algorithm results are tabulated

S.No	Pattern with size	No of times pattern occurred	Single processor Computing time		Two processors Computing time			Three Processors Computing time				Four Processors Computing time				
			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-(23bytes)	276	2	2	17	0	17	28	1	0	28	0	10	1	1	10
2	P2-(5bytes)	985	5	5	3	8	8	31	3	2	31	1	1	1	30	30
3	P3-(6bytes)	2347	4	4	8	2	8	10	6	2	10	2	1	1	10	10
Communication Time			0 milliseconds		9.67 milliseconds			12.27 milliseconds				15.92 milli seconds				

Figure .4: Integrated Algorithm Shows the variation of time among the processors for File 1

Table 2 shows the output results of file1 of 2 MB, for three patterns of different sizes P1 (23 bytes), P2 (5 bytes) and P3 (6 bytes). The table gives the understanding that the algorithm takes less time when number of processors increase and the time also varies with the pattern size. It is evident that, If pattern size increases and also No .of occurrences increases then the computing time reduces for given text file .The table gives the reading that the size of the pattern is proportional to the elapse time. But as the No. of processors increases, it gives the abnormal computing/search timings was observed (increasing and again decreasing).

The variation in timings for three search patterns are compared with the help of combined graph constructed using on line graph method shown above. In the above graph processor mentioned on x-axis computing time mentioned on y-axis in milliseconds. From the graph it is evident that three patterns are behaving abnormally, some times computing time increases as the No. of processors increases and again decreases was observed. Hence ingeneral we can conclude that integrated algorithm performance is not good when compared with hype algorithm. So hype algorithm is excellent for string matching in large text data bases, because it gives smart results and performances.

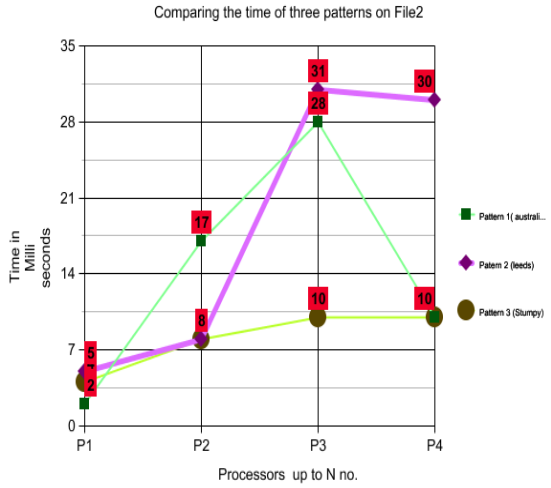

Figure 5 :On line graph for Comparison of three patterns for Text File 1.

6.3 Performances Comparison of Five Different Algorithms:

The experimental results presented in this paper which allows comparing of search /computing time of five different algorithms. Fast-search, BMH, Bi-Kun, Integrated and Hype Algorithms implemented in the JAVA programming language using RMI method. For simplicity reasons we experimented on only N= 4 No. of processors and are heterogeneous systems of each configuration is more than 650GB HDD, 6GB RAM and intel processor 2.5Ghz. The length of text is 2 MB and pattern file of size ranges from 2 bytes to 26 bytes and experiment is conducted almost 12 time I.e 2,4, 6, 8,10,12,14,16,18,20, 22 24 and 26 bytes, each time pattern size increasing to 2 bytes. The text file taken from TREC -05psn and each time pattern is selected randomly. The test results shows that the pattern size is small then fast-search and Bi-kun Algorithms are having better performance than BMH. As pattern size increases Bi-kun Algorithm is little better than BMH and fast-search algorithms. But in case of Hype and Integrated Algorithms the performances are improved to the better extent when compared with oher three algorithms. The search timings of all five Algorithms are tabulated in the table, and it is evident that The Hype algorithm works well and the performance is almost double when compare with Bi-kun algorithm. It also identified that for bigger pattern sizes the Hype algorithm

gives phenomenal growth in the performance and search time reduces further. In the other hand integrated algorithm gives the non-uniform performance.

Table .3 BMH,Fast Search , BI-KUN Integrated and Hype Algorithms test results are tabulated.

S.No	Pattern size in Alfbets (bytes)	BMHAlgorithm Computing time	Fast search Algorithm Computing time	BI KUN Algorithm Computing time	Integrated Algorithm Computing time	Hype Algorithm Computing time
1	2	100	45	42	25	16
2	4	55	38	33	22	14
3	6	38	33	26	20	13
4	8	35	28	25	18	12
5	10	30	25	24	16	10
6	12	28	24	20	16	10
7	14	26	22	20	14	9
8	16	24	20	19	13	8
9	20	24	20	18	13	7
10	22	24	20	18	12	6
11	24	24	20	18	12	5
12	26	24	20	18	12	5
Communication time		450ms	375ms	192 ms	17ms	10 ms

Figure .6: BMH,Fast Search , BI-KUN Integrated and Hype Algorithms Shows the variation of time among the processors for give Text File.

Table 3 shows the output results of file2 of 2MB, for 12 patterns of size ranges from 2bytes 26 bytes in increase of multiples of 2bytes each time. In the table the results of five algorithms called as BMH, Fast-search, BI-Kun and integrated & Hype algorithms are tabulated. The table gives comparison of five different algorithms and we can understand how the algorithms are behaving with different pattern sizes for given text file.It is evident that the pattern size increases the search time reduces for given text file was observed and it is also identified search/computing time is always inversely proportional to No. of processors involved. From the table we can easily conclude that the HYPE Algorithm exhibits the highest performance.

This graphs is constructed on line by feeding the results from the above table. The variation in time for twelve search patterns are compared with help of combined graphs constructed using online graph method shown above. It is evident that the pattern of size 2 bytes and for text file of size 2MB and No.of processor involved is four then , the BMH takes the 100 ms, Fast-search takes 45 ms, Bi-kun takes 42 ms , where as our own algorithms takes very less time when compare with other three algorithms, integrated algorithm takes 25 ms and Hype algorithm takes 16 ms. It is identified that, If No.of processors increase the search time reduces further and also it decreases drastically as the pattern size increases. Hence our experimental results give excellent out puts and we also conducted more experiments but, results are not presented due space problem , and it is discussed theoretically. From the table we can easily conclude that the HYPE Algorithm exhibits the highest performance.

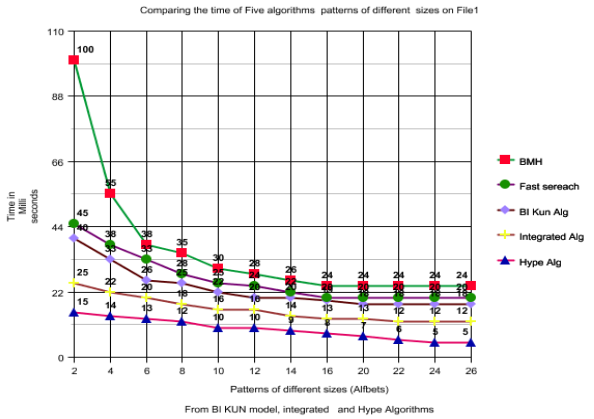

Figure .7 :On line graph for Comparison of BMH,Fast Search, BI-KUN Integrated and Hype Algorithms for different input patterns .

7. Conclusion

In this paper, we presented Hype and Integrated Algorithms on multi-processors in parallel environment called as hypercube network. And also we compared the performances of above algorithms with a practical distributed string matching algorithm developed by Bi-kun , which is suitable and efficient in distributed memory computing environment. We also presented an improved single string matching algorithm based on the Fast-Search algorithm proposed by Cantone and Faro and BMH algorithms. This distributed architecture is also suitable for paralleling the multi pattern string matching algorithms and approximate string matching algorithms. The experimental results shows that our Hype and Integrated Algorithms gives better performance over other Algorithms. 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 like image and video files etc.

Acknowledgment

At the outset I sincerely thank our guides Prof. S. Viswandha Raju and Prof. A govardhan for their never ending guidance and cooperation. I am also thankful to my dear students Mr. Mohmad Akram and Sana fathima for their hard work while developing this algorithms and writing this paper. I also extend my thanks to all who are directly or indirectly helped me to archive this. finally, I thank to my wife for understanding me and allowing me to do my PhD work day and night.

References

[1] S.ViswanadhaRaju, K.S.M.V.kumar,"Implementation of String Matching on Multiprocessors Using Divide and Conquer Technique " in 20113rd international Conference on Machine Learning and Computing (ICMLC) WWW.Elsevier.com,Dec2011.

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

[3] Roi Blanco ,B.Barla cambazoglu "8th LSDS-IR'10 –ACMSIGIR FORM,Vol 44 No.2 Dec-2010

[4] http://www.worldwidewebsize.com

[5] Cooks and Rivest "Two-way Pushdown Automata For String Matching"journal of IEEE.1977.

[6] www.lsdsir.org-page-id=21

[7] R.S. Boyer and J.S.Moore. Afast String Searching Algo-rithm Communications of the ACM, 20(10):762–772, 1977.

[8] KSMVKumar, S Viswanadharaju and A Govardhan "Overlapped Text Partion Algorithm For Pattern Matching On Hypercube Networked Model"Global Journal of Computer Sceince and Technology ,Vol 13, issue 4,April2013.

[9] A. V. Aho and M. J. Corasick. Efficient String Matching: An Aid to Bibliographic Search. Communicationof theACM,18(6):333–340, 1975.

[10] KSMVKumar,SViswanadharaju andAGovardhan "APartition Algorithm For Matching String Patterns In Large Databases" IJCSMR ,Vol 1, issue 2,sept 2012.

[11] M Allen, B. Wilkinson,"Parallel Programming: Techniques and Applications using Networked Workstations and Parallel Computers", Prentice Hall, 1999.

[12] 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

[13] N. Tuck, T. Sherwood, B. Calder and G. Varghese, "Deterministic memory-efficient string matching algorithms for intrusion detection," in *Proc. IEEE INFOCOM*, vol. 4, pp. 2628-2639, March 2004.

[14] 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.

[15] R.Sidhu's V.K Prasarna " Fast regular Expession Matching using FPGA's " In IEEE symposium on FPCCM; Rohnertpark,CA USA,April2001.

[16] Edward Frenandez, W Najjar and S Lonardi " String-matching in Hardware using the FM-Index"In IEEE1998.

[17] Snort. www.snort.org.

[18] S.Viswanadha Raju, S.R.Mantena, A.Vinayababu and GVSRaju, 2006, "Efficient Parallel String Matching Using Partition Method", Proc PDCAT-2006, IEEE Computer Society, 281-284.

[19] Herbert Scheldt's (Osborne) Java 2--Complete Reference, 5th edition-2008.

[20] RMI:http://java.sun.com/products/jdk/rmi/, http://www.ece.vill.edu/~khenry/rmiapp/

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

[22] Z. Galil, "Optimal parallel algorithms for string matching," in *Proc. 16thAnnu. ACM symposium on Theory of computing*, pp. 240-248, 1984.

[23] Hiroki Arimura,Atsushi waraki Ryoichi Fujino and Stno Arikawa " A Fast Algorithm for Discovering optimal String Parrens in Large Text Databases"

[24] Zheng Liu, Xin Chen, James Borneman and Tao Jiang, "A fast algorithm for approximate string matching on gene sequences," in *Symposium. 16th Annu. Combinatorial Pattern Matching, LNCS, Springer-Verlag*, vol.3537, pp. 79-90, June 2005.

[25] R. N. Horspool, "Practical Fast Searching in Strings," *Software - Practice and Experience*, vol. 10, pp. 501-506, 1980.

[26] D. Cantone and S. Faro, "Fast-Search: A new efficient variant of the Boyer-Moore string matching algorithm," in *Proc. Second International Workshop on Experimental and Efficient Algorithms, LNCS, Springer-Verlag*, Vol. 2647, pp. 47–58, May 2003.

[27] CHEN Guo-liang, LIN-Jie, and GU Nai-jie, "Design and analysis of string matching algorithm on distributed memory machine," *Journal of Software*, vol. 11, pp. 771-778, 2000.

[28] C. Charras and T. Lecroq, "Exact string matching algorithms," Laboratoire d'Informatique de Rouen Université de Rouen. Available: http://www-igm.univ-mlv.fr/~lecroq/string/

[29] U. Vishkin, "Optimal parallel matching in strings," *Information and control*, vol. 67, pp. 91-113, 1985.

[30] Bi Kun, Gu Nai-jie, Tu Kun, Liu Xiao-hu, and Liu Gang, "A Practical Distributed String Matching Algorithm Architecture and Implementation" in *Proc. World Academy of Science, Engineering and Technology 10 2007.*

K S M V Kumar did his Intermediate from Dr.G.G Degree college W.G Dt,AP during 1989-1991, B.Tech From Nagarjuna University,Guntur in 1997, and M.Tech from JNTUH Engineering college Hyderabad in 2003. Presently he is pursuing PhD From JNTUH ,Hyderabad, AP,INDIA**.** He worked as Asstitant .professor in GRIET, HYD.and he worked in anther college as an Associate Professor in cse department. Presently he is working as an professor in CSE department in SIET,HYD**.** Mr. Kumar is life time member for ACM , CSI and also Member in Various Editorial Boards like IJCSMR, IJEIT,IJESIT and IJCRT

S.Viswanadha Raju is a distinguished academician whose advanced research work in the field of Information Retrieval, Data Mining and Biometric Systems are globally recognized. After schooling and graduation he joined the JNTUniversity Hyderabad for his Postgraduate Program MTech (CSE) and subsequently Ph. D., from Acharya NagarjunaUniversity in Computer Science and Engineering in the area of Information Retrieval. He served as a Director of

MCA (Accredited by NBA) at Gokaraju Rangaraju Institute of Engineering and Technology (GRIET) during the period of June 2009 to July 2010 and preceding to this served as a Head of the Dept of CSE/MCA (CSE- twice accredited by NBA) at GRIET for a period of 7 Years. Presently he is HOD and Professor in the Department of CSE in JNTUH jagityal,AP,INDIA. S.V.Raju substantial and outstanding contribution to the field of academics in general and Computer Science and Engineering in particular has been duly recognized by National and International Organizations. To cite a few, he is the life member of IETE, life member of ISTE, life member of CSI and life member of IACSIT etc. His expertise is widely utilized by academic bodies such as NBA-AICTE, JNTUH and other universities.

A .Govardhan did his Intermedi from APRJC Nagarjuna Sagar, during 1986-1988, B.E in Computer Science and Engineering from Osmania University College of Engineering, Hyderabad in 1992, M.Tech from Jawaharlal Nehru University(JNU), Delhi in 1994 and he earned his Ph.D from Jawaharlal Nehru Technological University,Hyderabad He has been conferred A.P.State Best Teacher Award, Indian Glory Achievers Award, Mother Theresa Seva Ratna Award , Indira Gandhi Seva Ratna Award, Rajiv Gandhi Seva Ratna Award for the year 2012, CSI Chapter Patron Award (2010-2011), Shining Image of India Award , Seva Chakra Puraskar, Best Principal Award for the year 2011, Bharat Jyoti Award, Rajiv Gandhi Excellence Award, Best Citizens of India Award, Life Time Achievement Gold Medal Award, Rashtriya Vidya Saraswati Puraskar, Eminent Educationist Award for the year 2010 and Certificates of Excellence for Outstanding services, Dr. Govardhan is a member in Executive council,JNTUH,Member of Standing Committee for Academic Senate,Member of of Finance Committee and Member in Sports Council, JNT University Hyderabad. He is a Member on Board of Studies, JNT University Hyderabad, Adikavi Nannaya University,Rajahmundry, Satavahana University, Karimnagar and VR Siddhartha Engineering College, Vijayawada. He is the Vice-Chairman for CSI Hyderabad Chapter. He is a Member on the Editorial Boards for Eight International Journals