An Index based Pattern Matching using Multithreading

S. Nirmala Devi
Research Scholar
Bharath University
Chennai – 600 073. India

S.P. Rajagopalan
Professor Emeritus, Dr. M.G.R Educational and
Research Institution University
Chennai-600095. India

ABSTRACT
Pattern matching, the problem of finding sub sequences within a long sequence is essential for many applications such as information retrieval, disease analysis, structural and functional analysis, logic programming, theorem-proving, term rewriting and DNA-computing. In computational biology the essential components for DNA applications is the exact string matching algorithms. Many databases like GenBank were built by researchers for DNA and protein sequences; the string matching problem is the core problem for searching these databases. As the size of the database grows, the more important research area is to design an efficient string matching algorithms. This paper proposes a new pattern matching technique called An Index based Pattern matching using Multithreading for DNA sequences. The method specified in this paper performs parallel string searching using multiple threads simultaneously, each thread is responsible for searching one part of the text.. The proposed algorithm is an efficient algorithm that can be used to search for exact occurrences of patterns in DNA sequences.

Keywords
Exact String matching algorithms, pattern matching, DNA sequence, Multithreading, Context Switching.

1. INTRODUCTION
Pattern Matching is one of the important issues in the research areas of computer science. The field of bioinformatics has many applications in the modern world which includes text editors, Intrusion Detecting Systems, search engine, molecular medicine and Comparative biology etc., DNA - deoxyribonucleic acid which exists in chromosomes and mitochondria or chloroplast of cells contains the hereditary information of living things. DNA contains genetic instructions of an organism. The basic units of DNA are nucleotides and each nucleotide is one of the following four types: adenine (A), guanine (G), cytosine (C) and thymine (T). It can be viewed as a long sequences of A's, G's, C's and T's. It is very difficult to retrieve necessary information from the sequence when the size of the database grows.

Biologists are often interested in performing a simple database search to identify proteins or genes that contain a well-defined sequence pattern. Many algorithms have been developed each designed for a specific type of search. For Fast pattern matching techniques more efficient methods are required.

Let P={p1,p2,p3…pm } be a set of patterns which are strings of nucleotide characters from a fixed alphabet set called Σ = {A,C,G,T}. Let T be a long sequence of strings consists of characters in Σ denoted as Σ*. The problem of single pattern matching is to find all occurrences of pattern P in text T. If more than one pattern is matched against the given input text simultaneously, then it is called as multiple pattern matching.

Approximate String matching- Given a text string T of length n, a pattern string P of length m and a maximal number of errors allowed k, the approximate string matching is to find all text positions where the pattern matches the text up to k errors, where errors can be substituting, deleting, or inserting a character. For instance, if T = "pttapa", P = "patt" and k = 2, the substrings $T_{1,2}$, $T_{1,3}$, $T_{1,4}$ and $T_{5,6}$ are all up to 2 errors with P[1].

The main objective behind the pattern matching algorithms is to reduce the total number of character comparisons between the pattern and the text, reduce execution time and to increase the overall efficiency. The improvement in the efficiency of a search can be obtained by choosing the index value of the characters after each attempt.

We propose a new algorithm called An Index based Pattern matching using Multithreading for exact pattern-matching for DNA sequence. In this method, the characters in the input text is scanned from left to right till the end of the string for the first occurrence of pattern in Text and its position is retrieved.. In order to search for a particular pattern in the string T, the searching is based on the index value of the first character of the pattern P .If there is a mismatch, we skip the search and search starts on the next index of the first character of pattern P. This process is continued till the end of the String T. The efficiency of the new algorithm is supported by favorable experimental results obtained by comparison against prominent algorithms described in the next section.

2. BACKGROUND AND RELATED WORK
Several Algorithms have been proposed and they have their own advantages and limitations based on the text and pattern length. Generally there are two ways to search a pattern P against Text T.

Depending on the problem domain, most of the well-known algorithms [2] and [3] work in two phases.(i.e) ., preprocessing phase and the search phase. In the preprocessing phase, it processes the text and builds a data structure and this information in the search phase to reduce the total number of character comparisons and hence reduce

the overall execution time. Some algorithms performs search on text without preprocessing [4]. Such algorithms are called online search algorithms.

In the IKPMPM algorithm [5] a table is built called index table for all types of input patterns. For searching the pattern against the Text it uses the index table.

In the MSMPMA[8] the algorithm scans the input file to find all occurrences of the pattern based upon the skip technique.

GRASPm [9] algorithm improves exact pattern-matching in genomic sequences. GRASPm could be classified as a heuristic-based algorithm, analyzing multiple pattern alignments within a wide search window and using a novel filtering heuristic to maximize efficiency.

This proposed method performs preprocessing to get the index. By using this index as the starting point of matching, it compares the Text contents from the defined point with the pattern contents.

2.1 Thread and Multithreading Motivation

A problem with single-threaded applications is that lengthy activities must complete before other activities can begin (actions execute one after another.) In a multithreaded application, Multithreading [6] allows two parts of the same program to run concurrently.

Java is unique among popular general-purpose programming languages in that it makes concurrency primitives available to the applications programmer. The programmer specifies that applications contain threads of execution, each thread designating a portion of a program that may execute concurrently with other threads. Multithreading gives the Java programmer powerful capabilities that are not available in C and C++, the languages on which Java is based [6].

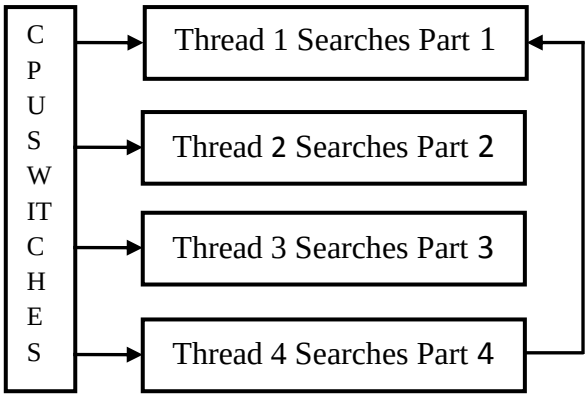


Fig 1: CPU Context Switching between Threads

Multithreading allows a program or a process to execute many tasks concurrently (at the same time and parallel). It allows a process to run its tasks in parallel mode on a single processor system [7]. CPU performs context switching between threads and it seems that threads are executed at the same time. This study proposes a multithreading text search approach to improve search performance at a single CPU machine. The idea is to have multiple threads that search the text from different positions. The pattern may occur at any position, having more than one search is better than searching the text sequentially from the first character to the last one.

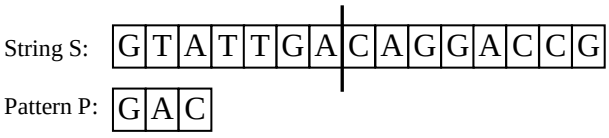
The first thread examines the first character of its assigned text part, CPU makes a context switch specified in Fig.1 to the second thread to check the first character of its part and so on. This process is repeated until the whole text is examined by all the threads.

3. AN INDEX BASED PATTERN MATCHING USING MULTITHREADING

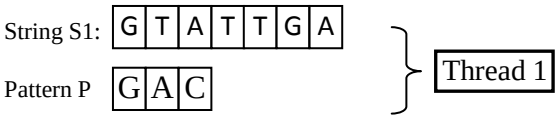
In this proposed method a very large size DNA sequence is divided into parts depending upon the pattern size. The main idea by using multithreading is to solve the pattern matching problem on a single CPU machine is to have multi search threads that searches the pattern simultaneously in a timesharing manner. From different positions the threads starts searching for the pattern, the speed of finding the required pattern will increase.

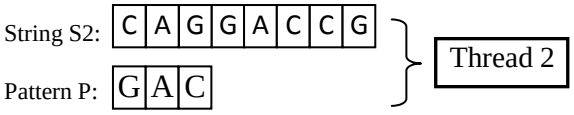
When splitting the Text into parts problem will not occur if a match is found entirely in the first half or in the second half of the string.

Situations may occur when a part of pattern arises in one string and another part of pattern arises in the consecutive string, then we have to make sure about that the pattern matches or not.

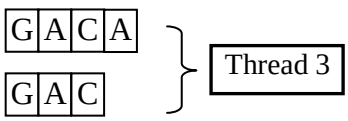


Split S into S1 and S2





If the pattern size is "m" then we will store the last (m-1) elements of string1 and the first (m-1) elements of consecutive string2. So total of 2(m-1) elements will be stored



in a new derived string. Now the pattern of size "m" will match its elements in this new derived string.

String S3:

Pattern P:

It then shows number of attempts and the occurrence of pattern if exist in the same way. This process will be done for all joining parts of the strings. Now we will combine all the threads execution result together.

3.1 Algorithm

Input : String T of n characters and a pattern P of m characters, where S, P belongs to as Σ^{*}.

Output : The number of occurrences and number of characters compared.

Step 1 : [initialization of variable]

String array s[x], n_occ:=0, cmp:=0,index=0;

Step 2 : assigning values for

s[0]←-S.substring(0,n/2-1), s[1]←S.substring(n/2,n),

s[2]←S.substring(n/2-(m-1),n/2+(m-1))

Step 3 : creating multiple threads and the run method performs preprocessing and search for the pattern in the portion of the text allotted to each thread.

Searching phase

Step 4 : char start_char_pattern=P.charAt(0);

For i:=index;i<size;i++

Index= T.indexOf(start_char_pattern,index);

j:=starting index of pattern P

cmp:=cmp+1

Step 5 : while (j<m) && (P.charAt(j)==T.chatAt(ind+j)

j:=j+1;

if(j==m)

n_occ:=n_occ+1;

End If

i=index+j;

End while

End For

Step 6: print "Number of occurrences – n_occ,Number of comparisons cmp".

3.2 Working Example.

The genome databank consists of gene sequences(NCBI site http://www.ncbi.nlm.nih.gov/nuccore/1762443 ?report=fasta) .Full Sequence in Fasta Format.>

>gi|1762443|gb|U60816.1|HSSLC07 Human cystine transporter rBAT (SLC3A1) gene, exon 7

CCCCGATGACACTGAACCTTGTCAACTCTTATAGGTT
CATGGGGACTGAAGCCTATGCAGAGAGTATTGACAG
GACCGTGATGTACTATGGATTGCCATTTATCCAAGA
AGCTGATTTTCCCTTCAACAATTACCTCAGCATGCTA
GACACTGTTTCTGGGAACAGCGTGTATGAGGTTATC
ACATCCTGGATGGAAAACATGCCAGAAGGAAAATG
GCCTAACTGGATGGTAAGTTCTCATGACAGCAGAGT
AAGGAGAGGACAGCGAT

To validate the proposed method, a part of the gene sequence (only 31 residues from a gene sequence has been used(see below for details)

Let us take a string S=G T A T T G A C A G G A C C G T G A T G T A C T A T G G A C T of 31 characters and P=G A C.

n=31 and m=3.

The length of String S is 31. n/2 of S is 15 & 16

The String S is split into 3 substrings S1,S2, S3 as S1 contains character from 0 to 14 and S2 from 15 to 30 and S3 contains character based on pattern length. It contains the last m-1 values of S1 and first m-1 values of S2(values from 13th location to 16th location of S).Threads are created based on the number of substrings and passed as argument to the thread. The run method does the searching process.

The algorithm first finds the index of the first occurrence of the first character of pattern in the text.

Based on that search begins from left to right.

S1=G T A T T G A C A G G A C C G

P= G A C

The first character matches then it compares the second characters of pattern and text. If it matches it proceeds the same for the remaining characters based on pattern length. If it does not match it finds the next index of first character of pattern in text.

S1=G T A T T G A C A G G A C C G

P = GAC

Compares the second character against the character in text S1.

S1=G T A T T G A C A G G A C C G

P = G A C

Compares the third character against the character in text S1.

S1=G T A T T G A C A G G A C C G

P = G A C

Now all the character matches it prints the message the pattern found at the location 5 in the string. It follows the same procedure for the remaining characters in the string s1 and for the other threads.

Experimental results of the proposed algorithm with different Patterns, the number of occurrences and the number of character comparisons for the part of the string of Human cystine transporter rBAT (SLC3A1) gene, exon 7 (Full sequence in Fasta Format). We have implemented and tested the **Index Based Pattern Matching using Multithreading** algorithm for different patterns matching using object oriented programming with Java and the results are shown in the following Table 1..

Table 1. Experimental Results of Index Based Pattern Matching using Multithreading algorithm

Patterns (P)	Size of P	No.of Occur	No.of comparision
A	1	8	8
GA	2	4	16
GAC	3	3	22
GGAC	4	2	18
ATTGA	5	1	19
GATGTA	6	1	19
GACAGGA	7	1	19

The operating system's task scheduler allocates execution time to multiple tasks. By quickly switching among executing tasks, it creates the impression that the tasks execute simultaneously. If it didn't switch among the tasks, they would execute sequentially.

Few threads on One CPU may increase performance in case you continue with another thread instead of waiting for I/O bound operation and the threads share a single memory space.

In molecular biology this type of large sequences are common to compare with other sequences. To check whether the given pattern presents in the sequences or not we need an efficient algorithm which searches in less time. There are many algorithms which perform the search and each have its advantages and disadvantages. This algorithm will decrease the number of comparisons.

4. RESULTS AND DISCUSSION

The results were compared with Brute-Force, Not So Naïve and Morris-Pratt algorithm and are shown in the table 2 and also plotted in the graph as shown in the Fig. 2.

Table 2. Comparisons of different algorithms with Thread

Pattern	No. of chars	Number of character comparisons			
		Index Based	Brute Force	Not SoNaive	Morris -Pratt
GA	2	132	336	269	309
GAC	3	166	362	281	329
GGAC	4	153	360	207	320
ATTGA	5	186	373	283	344
GATGTA	6	168	367	297	328
GGCCTAA	7	157	350	197	327

Table 2 shows the comparison between different existing algorithms with the proposed technique in terms of number of comparisons. A size of 270 characters of Human cystine transporter rBAT (SLC3A1) DNA gene is given as Input for the existing and the proposed method and 6 different random patterns from the above shown DNA data set and the pattern size starting from 2 character to 7 characters chosen from the DNA dataset.

Fig 2. shows the comparisons of different algorithms with the proposed technique. The current technique gives good performance in reducing the number of character comparisons compared with other popular methods. The dotted line shows the proposed model whereas Brute-Force, Not so naïve and Morris Pratt are shown by solid lines. Towards X-axis we have taken randomly different pattern sizes range from 2 to 7 whereas towards Y-axis shows the total number of comparisons.

This approach is efficiently improving the time complexity as it is based on multithreading technique. The use of pattern matching is very broad and efficient pattern matching algorithm can improve system performance. This Multithreaded implementation improves the CPU utilization and increases the time efficiency.

5. CONCLUSION

We presented Index Based Multithreaded Pattern matching algorithm with a simple logic which is very easy to implement. This method can also be used for pattern matching in protein sequences and also for English Text. This algorithm depends on the pre-processing phase which retrieves the index.

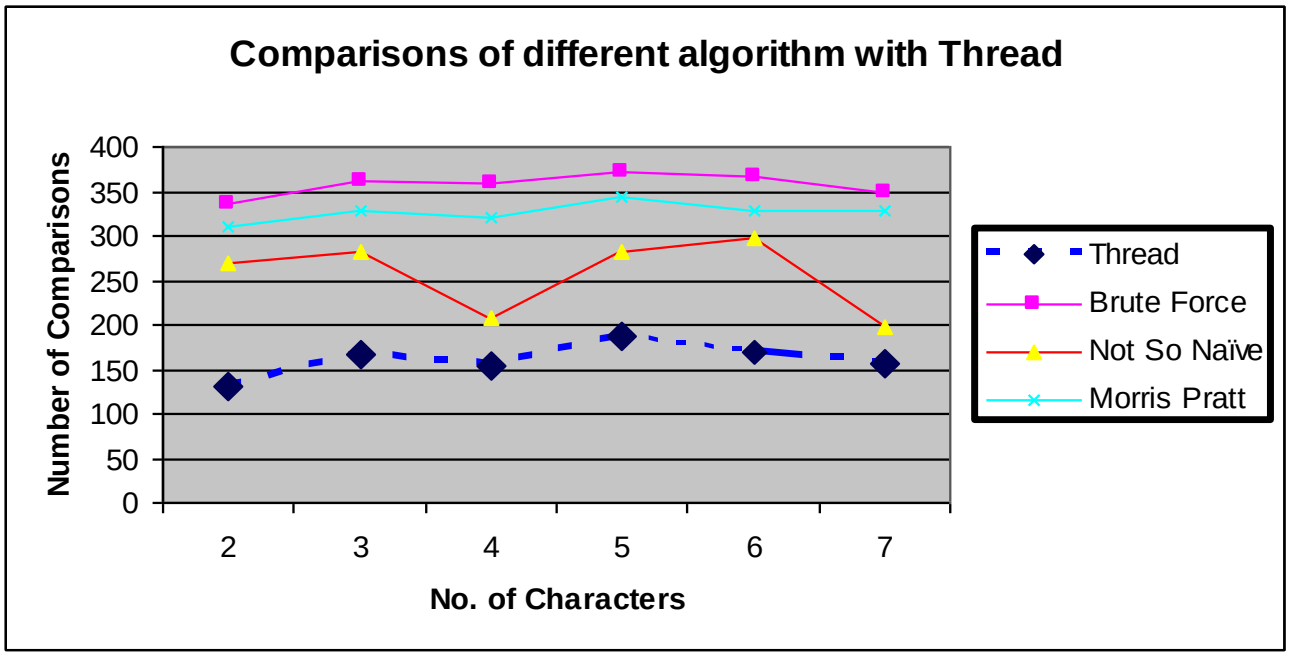


Fig 2. Comparison of Different Algorithms with Index Based Thread

We have taken the input of 270 characters and tested randomly by taking different pattern sizes. Our further interests are focused for developing algorithms for multiple pattern matching and patterns that include regular expressions to meet the challenges and demands put forward by the present-day computational genomics research. Multithreaded implementation improves the CPU utilization and this approach provides best performance related to DNA sequence dataset.

6. REFERENCES

[1] Dan Gusfield, Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology, Cambridge University Press, New York, 1997.

[2] Knuth D., Morris.J Pratt.V Fast pattern matching in strings, SIAM journal on computing.

[3] Boyer R. ande Moore J., "A Fast String Searching Algorithm,"Computer Journals of Communications of the ACM, vol.20, no.10, pp.762-772, 1997.

[4] Christian Charras, "Brute Force algorithm. http://www-igm.univmlv-fr/~lecroq/string/node3.html#SECTION0030..

[5] Raju Bhukya,DVLN Somayajulu,"An Index Based K-Partitions Multiple Pattern matching Algorithm", Proc.of Int.Conf. on Adfvances in Computer Science 2010.

[6] Deitel P. and Deitel H., Java How to Program, Prentice Hall, 2003.

[7] T. Ungerer,," A Survey of Processors with Explicit Multithreading", ACM Computing Surveys, Vol. 35, No. 1, March 2003, pp. 29–63.

[8] Ziad A.A. Alqadi, "Multiple Skip Multiple Pattern Matching Algorithm ", IAENG International Journal of Computer Science, 34:2, IJCS_34_2_03, Advance online publication: 17 November 2007.

[9] Deusdado, S. and Carvalho, P.(2009) 'GRASPm: an efficient algorithm for exact pattern-matching in genomic sequences', Int J. Bioinformatics Research and Applications, Vol.