Global Journal of Computer Science and Technology
Hardware & Computation
Volume 13 Issue 1 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

Parallel String Matching with Multi Core Processors-A Comparative Study for Gene Sequences

By Chinta Someswara Rao, K. Butchi Raju & Dr. S. Viswanadha Raju
Andhra University, India

Abstract - The increase in huge amount of data is seen clearly in present days because of requirement for storing more information. To extract certain data from this large database is a very difficult task, including text processing, information retrieval, text mining, pattern recognition and DNA sequencing. So we need concurrent events and high performance computing models for extracting the data. This will create a challenge to the researchers. One of the solutions is parallel algorithms for string matching on computing models. In this we implemented parallel string matching with JAVA Multi threading with multi core processing, and performed a comparative study on Knuth Morris Pratt, Boyer Moore and Brute force string matching algorithms. For testing our system we take a gene sequence which consists of lacks of records. From the test results it is shown that the multicore processing is better compared to lower versions. Finally this proposed parallel string matching with multicore processing is better compared to other sequential approaches.

Keywords : *string matching; parallel string mathing; computing model, DNA, multicore processing.*

GJCST-A Classification : *B.7.1*

Strictly as per the compliance and regulations of:

Parallel String Matching with Multi Core Processors-A Comparative Study for Gene Sequences

Chinta Someswara Rao [α], K Butchi Raju [σ] & Dr. S. Viswanadha Raju [ρ]

Abstract - The increase in huge amount of data is seen clearly in present days because of requirement for storing more information. To extract certain data from this large database is a very difficult task, including text processing, information retrieval, text mining, pattern recognition and DNA sequencing. So we need concurrent events and high performance computing models for extracting the data. This will create a challenge to the researchers. One of the solutions is parallel algorithms for string matching on computing models. In this we implemented parallel string matching with JAVA Multi threading with multi core processing, and performed a comparative study on Knuth Morris Pratt, Boyer Moore and Brute force string matching algorithms. For testing our system we take a gene sequence which consists of lacks of records. From the test results it is shown that the multicore processing is better compared to lower versions. Finally this proposed parallel string matching with multicore processing is better compared to other sequential approaches.

Keywords : string matching; parallel string mathing; computing model, DNA, multicore processing.

I. Introduction

The crisis of finding exact or non-exact occurrences of a pattern P in a text T over some alphabet is a central difficulty of combinatorial string matching and has a variety of applications in many areas of computer science [1-3]. String searching algorithms can be accomplished in two ways:

1. Exact match, meaning that the passages returned will contain an exact match of the key input.
2. Approximate match, meaning that the passage will contain some part of the key word input [4-6].

Although the dramatic development of processor technology and other advances have reduced search response to negligible times, string matching problem still remains a useful area of research and development for a number of reasons. Initially, as the size of data continues to grow, sequence searches will become increasingly taxing on search engines. Secondly, the pattern matching still remains an integral part of faster matching algorithms, typically comprising the final part of a search. Finally, researchers have to understand the classical methods of pattern matching to develop new efficient algorithms [7-10].

With the developments of new string matching techniques, efficiency and speed are the main factors in deciding among different options available for each application area. Each application area has certain special features that can be used by string matching technique best suited for that area [11-13]. This study implements a multithreading text searching approach to improve text searching performance at a multicore processing. The idea is to have more than one searcher thread that search the text from different positions. Since the required pattern may occur at any position, having multiple searchers is better than searching the text sequentially from the first character to the last one.

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

II. Related Work

Now a day's information retrieval attracted by the many researchers because of their importance in IT industry. So, this many researchers worked on this area since several years. In this paper we propose some of the techniques comparisons with multicore processing. In this section we discuss some previous techniques proposed by several authors' later section we will discuss about our actual procedure.

S.V. Raju et.al [14] proposes about grid computing in parallel string matching. Grid computing provides solutions for various complex problems. The function of the grid is to parallelize the string matching problem using grid MPI parallel programming method or loosely coupled parallel services on Grid. Parallel applications fall under three categories namely Perfect parallelism, Data parallelism and Functional parallelism, use data parallelism, and it is also called Single

Author α : Assistant Professor, Dept of CSE, SRKR Engineering College, Bhimavaram, AP, India.
E-mail : chinta.someswararao@gmail.com
Author σ : Associate Professor, Department of CSE, GRIET, Hyderabad, AP, India.
Author ρ : Professor & HOD, Department of CSE, Kondagattu, Jagithyal, A.P., India.

Program Multiple Data (SPMD) method, where given data is divided into several parts and working on the part simultaneously.

Perfect Parallelism

Also known as embarrassingly parallel. An application can be divided into sets of processes that require little or no communication.

Data Parallelism

The same operation is performed on many data elements simultaneously. An example would be using multiple processes to search different parts of a database for one specific query.

Functional parallelism: Often called control parallelism. Multiple operations are performed simultaneously, with each operation addressing a particular part of the problem.

Result

Here it shows the performance of string matching algorithms namely execution-time and speedup improved.

HyunJin Kim and Sungho Kang [15] propose an algorithm that partitions a set of target patterns into multiple subgroups for homogeneous string matchers. Using a pattern grouping metric, the proposed pattern partitioning makes the average length of the mapped target patterns onto a string matcher approximately equal to the average length of total target patterns. The target architecture is based on a memory-based string matching with homogeneous string matchers. In a string matcher, N homogeneous finite state machine (FSM) tiles are contained. An FSM tile contains a maximum of s states and takes n bits of one character at each cycle. Target patterns are distributed and mapped onto C string matchers. Each state has $2n$ pointers for the next state based on an n-bit input.

Result

By adopting the pattern grouping metric, the proposed pattern group partitioning decreases the number of adopted string matchers by balancing the numbers of mapped target patterns between string matchers.

Daniel Luchaup, et.al [16] they propose a method to search for arbitrary regular expressions by scanning multiple bytes in parallel using speculation. They break the packet in several chunks, opportunistically scan them in parallel, and if the speculation is wrong, correct it later. They present algorithms that apply speculation in single-threaded software running on commodity processors as well as algorithms for parallel hardware.

Result

It is a speculative pattern matching method which is a powerful technique for low latency regular-expression matching. The method is based on three important observations. The first key insight is that the serial nature of the memory accesses is the main latency-bottleneck for a traditional DFA matching. The second observation is that a speculation that does not have to be right from the start can break this serialization The third insight, which makes such a speculation possible, is that the DFA-based scanning for the intrusion detection domain spends most of the time in a few hot states.

Hyun Jin Kim, et.al [17] propose a memory-efficient parallel string matching scheme. In order to reduce the number of state transitions, the finite state machine tiles in a string matcher adopt bit-level input symbols. Long target patterns are divided into sub patterns with a fixed length; deterministic finite automata are built with the sub patterns. Using the pattern dividing, the variety of target pattern lengths can be mitigated, so that memory usage in homogeneous string matchers can be efficient.

Result

The proposed DFA-based parallel string matching scheme minimizes total memory requirements. The problem of various pattern lengths can be mitigated by dividing long target patterns into sub patterns with a fixed length. The memory-efficient bit-split FSM architectures can reduce the total memory requirements. Considering the reduced memory requirements for the real rule sets, it is concluded that the proposed string matching scheme is useful for reducing total memory requirements of parallel string matching engines.

Charalampos S, et.al[18] they proposes that Graphics Processing Units (GPUs) have evolved over the past few years from dedicated graphics rendering devices to powerful parallel processors, outperforming traditional Central Processing Units (CPUs) in many areas of scientific computing. The use of GPUs as processing elements was very limited until recently, when the concept of General-Purpose Computing on Graphics Processing Units (GPGPU) was introduced. GPGPU made possible to exploit the processing power and the memory bandwidth of the GPUs with the use of APIs that hide the GPU hardware from programmers. This paper presents experimental results on the parallel processing for some well known on-line string matching algorithms using one such GPU abstraction API, the Compute Unified Device Architecture (CUDA).

Result

In this, both the serial and the parallel implementations were compared in terms of running time for different reference sequences, pattern sizes and number of threads. It was shown that the parallel implementation of the algorithms was up to 24x faster than the serial implementation, especially when larger text and smaller pattern sizes where used. The performance achieved is close to the one reported for

similar string matching algorithms. In addition, it was discussed that in order to achieve peak performance on a GPU, the hardware must be as utilized as possible and the shared memory should be used to take advantage of its very low latency. Future research in the area of string matching and GPGPU parallel processing could focus on the performance study of the parallel implementation of additional categories of string matching algorithms, including approximate and two dimensional string matching.

Thierry Lecroq[19] propose a very fast new family of string matching algorithms based on hashing q-grams. The new algorithms are the fastest on many cases, in particular, on small size alphabets. The string matching problem consists in finding one or more usually all the occurrences of a pattern $x = x[0..m-1]$ of length m in a text $y = y[0..n-1]$ of length n. It can occur, for instance, in information retrieval, bibliographic search and molecular biology.

Result

In this article they presented simple and though very fast adaptations and implementations of the Wu–Manber exact multiple string matching algorithm to the case of exact single string matching algorithm. Experimental results show that the new algorithms are very fast for short patterns on small size alphabets comparing to the well known fast algorithms using bitwise techniques. The new algorithms are also fast on long patterns (length 32 to 256) comparing to algorithms using an indexing structure for the reverse pattern (namely the Backward Oracle Matching algorithm). This new type of algorithm can serve as filters for finding seeds when computing approximate string matching.

Derek Pao, et.al [20] proposes that a memory-efficient hardware string searching engine for antivirus applications is presented. The proposed QSV method is based on quick sampling of the input stream against fixed-length pattern prefixes, and on-demand verification of variable-length pattern suffixes. Patterns handled by the QSV method are required to have at least 16 bytes, and possess distinct 16-byte prefixes. The latter requirement can be fulfilled by a preprocessing procedure. The search engine uses the pipelined Aho-Corasick (P-AC) architecture developed by the first author to process 4 to 15-byte short patterns and a small number of exception cases. Our design was evaluated using the Clam AV virus database having 82,888 strings with a total size that exceeds 8 MB. In terms of byte count, 99.3 percent of the pattern set is handled by the QSV method and 0.7 percent of the pattern set is handled by P-AC. A pattern with distinct 16-byte prefix only occupies up to three lookup table entries in QSV. The overall memory cost of our system is about 1.4 MB, i.e., 1.4 bit per character of the ClamAV pattern set. The proposed method is memory-based, hence, updates to the pattern set can be accommodated by modifying the contents of the lookup tables without reconfiguring the hardware circuits.

Hassan Ghasemzadeh[21] proposes that Mobile sensor-based systems are emerging as promising platforms for healthcare monitoring. An important goal of these systems is to extract physiological information about the subject wearing the network. Such information can be used for life logging, quality of life measures, fall detection, extraction of contextual information, and many other applications. Data collected by these sensor nodes are overwhelming, and hence, an efficient data processing technique is essential.

Result

Results show the effectiveness of this approach, both for reliable movement classification and reduction of communication.

HyunJin Kim and Seung-Woo Lee [22] propose a memory-based parallel string matching engine using the compressed state transitions. In the finite-state machines of each string matcher, the pointers for representing the existence of state transitions are compressed. In addition, the bit fields for storing state transitions can be shared. Therefore, the total memory requirement can be minimized by reducing the memory size for storing state transitions

Result

This letter proposed a memory-efficient parallel string matching engine in DFA-based string matching. The proposed string matcher can reduce the memory size for storing the existence of state transitions. In addition, the memory requirements can be reduced by sharing state transitions in the transition table. Considering the experiment results, it is evident that the proposed architecture is useful for reducing the storage cost of the DFA-based string matching engine.

Ali Peiravi and Mohammad Javad Rahimzadeh[23] proposes that String matching is a fundamental element of an important category of modern packet processing applications which involve scanning the content flowing through a network for thousands of strings at the line rate. To keep pace with high network speeds, specialized hardware-based solutions are needed which should be efficient enough to maintain scalability in terms of speed and the number of strings. In this paper, a novel architecture based upon a recently proposed data structure called the Bloomier filter is proposed which can successfully support scalability. The Bloomier filter is a Compact data structure for encoding arbitrary functions, and it supports approximate evaluation queries. By eliminating the Bloomier filter's false positives in a space efficient way, a simple yet powerful exact string matching architecture is proposed that can handle several thousand Strings at high rates and is amenable to on-chip realization. The proposed scheme is implemented

in reconfigurable hardware and compare it with existing solutions. The results show that the proposed approach achieves better performance compared to other existing architectures measured in terms of throughput per logic cells per character as a metric.

In this paper, we use parallel algorithms with multicore processors because with multicore processors we can increase the efficiency and the performance.

III. Computing Model with Multicore Porcessing

As personal computers have become more prevalent and more applications have been designed for them, the end-user has seen the need for a faster, more capable system to keep up. Speedup has been achieved by increasing clock speeds and, more recently, adding multiple processing cores to the same chip. Although chip speed has increased exponentially over the years, that time is ending and manufacturers have shifted toward multicore processing. However, by increasing the number of cores on a single chip challenges arise with memory and cache coherence as well as communication between the cores. Coherence protocols and interconnection networks have resolved some issues, but until programmers learn to write parallel applications, the full benefit and efficiency of multi core processors will not be attained [24-27].

IV. Proposed System Architecture

a) System Architecture

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

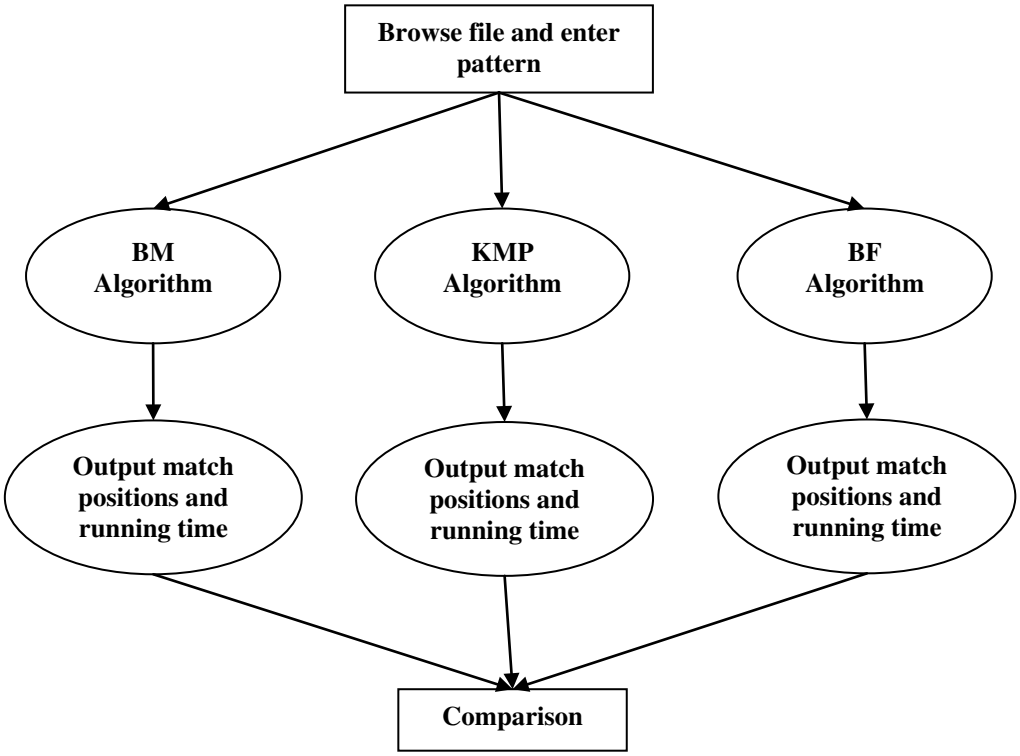

Figure 1 : Existing System

In the existing string matching architecture we search the required pattern sequentially at first we pass the required that is to be searched and this pattern is searched by using the three algorithms Brute force, KMP, Boyer Moore the entire string is passed through all the algorithms and the output match and the running time is calculate for the required pattern from all the algorithms and the algorithm with the least running time is selected, all this is done sequentially which takes more time to execute to improve the efficiency and the performance in this we use the parallel string matching algorithms with multicores processors as shown in Fig 2.

The proposed system Architecture of Comparison of parallel String Matching Algorithms is as follows in the below diagram. In this search the pattern parallel. in this at first we take the input as a string or text. The required text that is to be searched is further divided into further small patterns and all this patterns are passed on the different parallel algorithms like KMP

boyar Moore, brute force and at all the output position match and running time of all the patterns is calculated and the all the patterns of same algorithm are added and all the resulted running time are compared with other algorithms resulting time and from them the best one is taken as the efficient algorithm for the string matching.

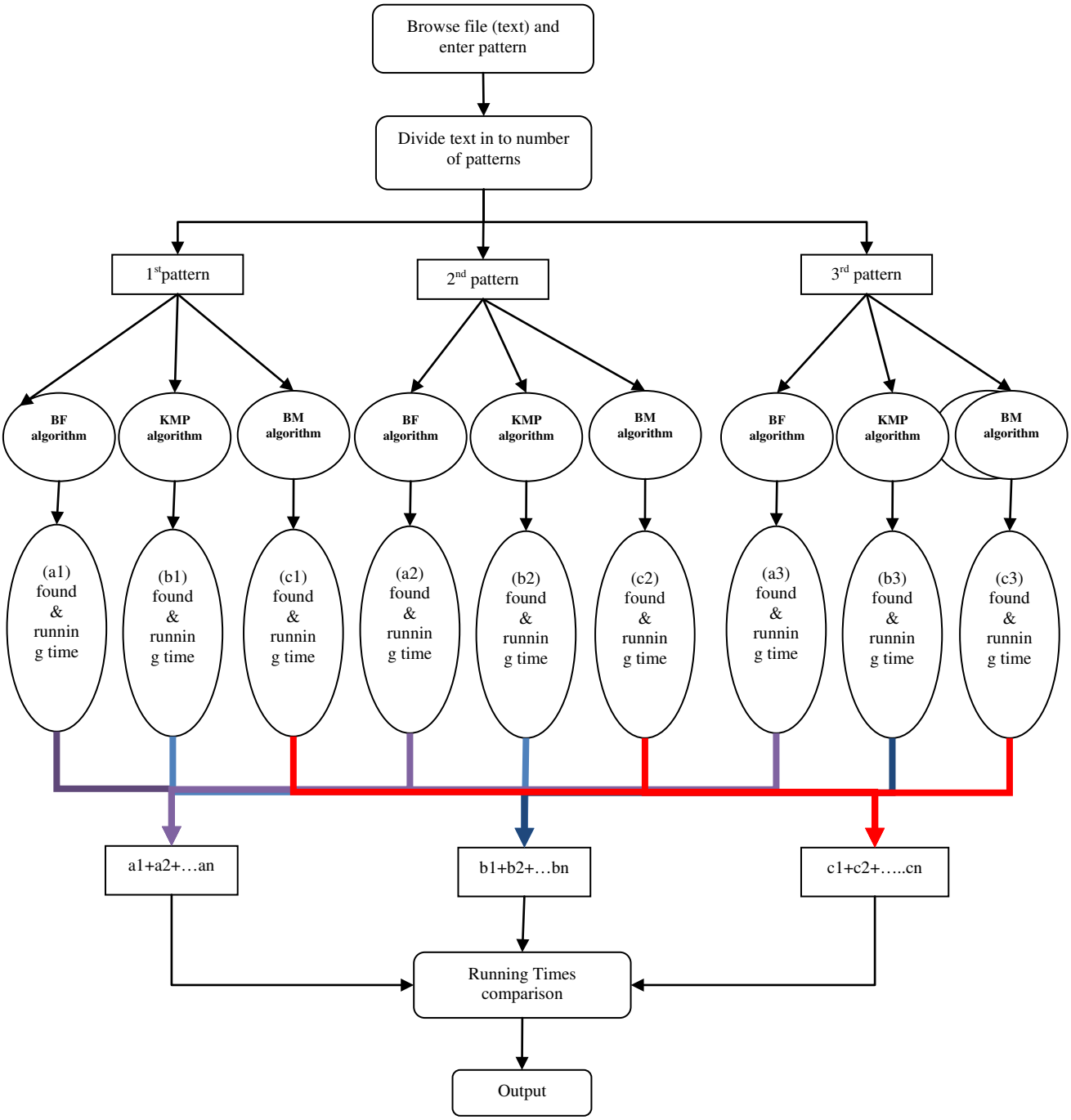

Figure 2 : Proposed Systems

V. Proposed Approach

In now a day as the current free textual database is growing vast there is a problem of finding the pattern by string matching the efficiency is decreased and takes more time. In our paper, we use parallel algorithms to increase the efficiency on multicore processor we pass the same string to all the three algorithms and we select the best based on the running time.

a) Implementation

Here we have to implement the proposed system with JAVA 1.7 multi threading, initially we have to implement the BF, KMP, and BM sequentially and then go for parallel implementation with threading on Multicore processor. Here we discuss some of them.

i. *Brute force Algorithm (BF) description and Implementation with parallel programming[28-30]*

The brute force algorithm consists in checking, at all positions in the text between 0 and n – m. whether an occurrence of the pattern starts there or not. Then, after each attempt, it shifts the pattern by exactly one position to the right. The brute force algorithm requires no preprocessing phase, and a constant extra space in addition to the pattern and the text. During the searching phase the text character comparisons can be done in any order. The algorithm can be designed to stop on either the first occurrence of the pattern, or upon reaching the end of the text. This code was run parallel in multiple threads to achieve good efficiency searching, which is shown in Table 1.

Table 1 : Pseudo code for BF

```
FileInputStream fstream = new FileInputStream("F:/multi/genesequence.txt");
DataInputStream in= new DataInputStream(fstream);
BufferedReader br = new BufferedReader(new InputStreamReader(in));
time = System.currentTimeMillis();
while ( ((str = br1.readLine()) != null)&&(i<=i1) ){
BruteForceSearch bfs = new BruteForceSearch();
String pattern = "AAGG";
bfs.setString(str, pattern);
first_occur_position = bfs.search();
System.out.println("The text '" + pattern + "' is first found after the " + first_occur_position + "
position.");
i++;}
time = System.currentTimeMillis() - time;
System.out.println("Time elapsed"+time);
```

ii. *Knuth Morris Pratt description and Implementation with parallel programming [28-30]*

Consider an attempt at a left position j, that is when the window is positioned on the text factor y[j .. j+m-1]. Assume that the first mismatch occurs between x [i] and y [i+j] with $0 < i < m$. Then, x[0 .. i-1] = y[j .. i+j-1]=u and $a = x[i] \neq y[i+j]=b$. When shifting, it is reasonable to expect that a prefix v of the pattern matches some suffix of the portion u of the text. Moreover, if we want to avoid another immediate mismatch, the character following the prefix v in the pattern must be different from a. The longest such prefix v is called the *tagged border* of u. This code was run parallel in multiple threads to achieve good efficiency searching, which is shown in Table 2.

Table 2 : Pseudo code for KMP

```
FileInputStream fstream = new FileInputStream("F:/multi/genesequence.txt");
DataInputStream in = new DataInputStream(fstream);
BufferedReader br = new BufferedReader(new InputStreamReader(in));
time = System.currentTimeMillis();
while ( ((str = br1.readLine()) != null) && (i<=i1) ) {
KMPS kmp = new KMPS();
String pattern = "AAGG";
kmp.setString(str, pattern);
first_occur_position = kmp.search();
System.out.println("The text '" + pattern + "' is first found after the " + first_occur_position + " position.");
i++;}
time = System.currentTimeMillis() - time;
System.out.println("Time elapsed"+time);
```

iii. *Boyer Moore Algorithm description and Implementation with parallel programming[28-30]*

The algorithm scans the characters of the pattern from right to left beginning with the rightmost one. In case of a mismatch (or a complete match of the whole pattern) it uses two precomputed functions to shift the window to the right. These two shift functions are called the *good-suffix shift* (also called matching shift) and the *bad-character shift* (also called the occurrence shift). This code was run parallel in multiple threads to achieve good efficiency searching, which is shown in Table 3.

Table 3 : Pseudo code for BM

```
FileInputStream fstream = new FileInputStream("F:/multi/genesequence.txt");
DataInputStream in = new DataInputStream(fstream);
BufferedReader br = new BufferedReader(new InputStreamReader(in));
time = System.currentTimeMillis();
while ( ((str = br1.readLine()) != null) && (i<=i1) ){
BoyerMoore bms = new BoyerMoore();
String pattern = "AAGG";
bms.setString(str, pattern);
first_occur_position = bms.search();
System.out.println("The text '" + pattern + "' is first found after the " + first_occur_position + " position.");
i++;}
time = System.currentTimeMillis() - time;
System.out.println("Time elapsed-in thread-1"+time);
```

b) Claims

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

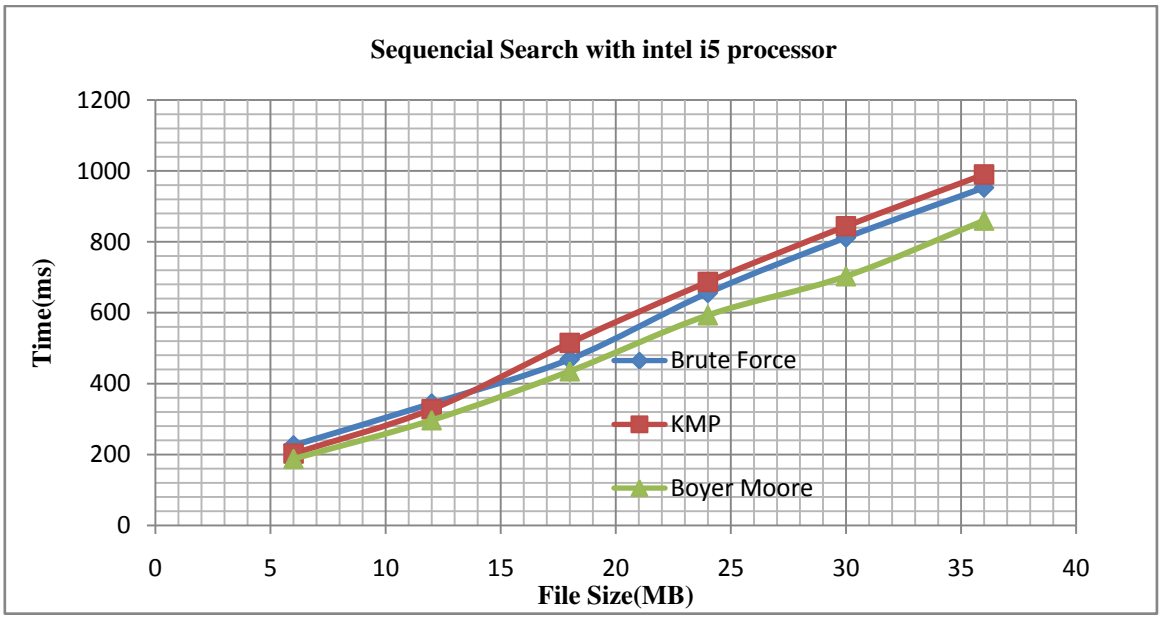

Figure 3 : Graph for sequential search (Time vs File size)

The Fig 4 shows(Graph)Execution time vs File size on parallel search with intel i5 processor using Boyer Moore, Brute force, KMP Algorithm. This graph shows the performance difference between Boyer Moore, Knuth Morris Pratt and Brute force algorithms. From the graph we clearly observe that BM is better compared to other approaches. The Fig 5 shows(Graph) Execution time vs File size using Brute force Algorithm on parallel search and sequential search with intel i5 processor. From the graph we clearly observe that parallel is much better than sequential search.

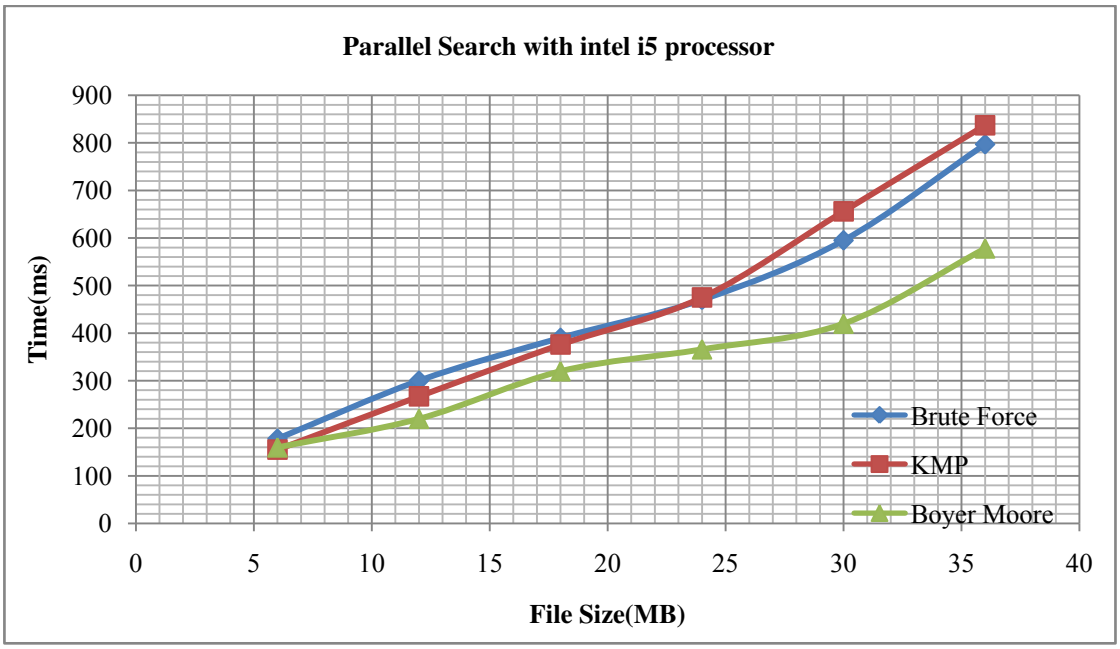

Figure 4 : Graph for parallel search (Time vs File size)

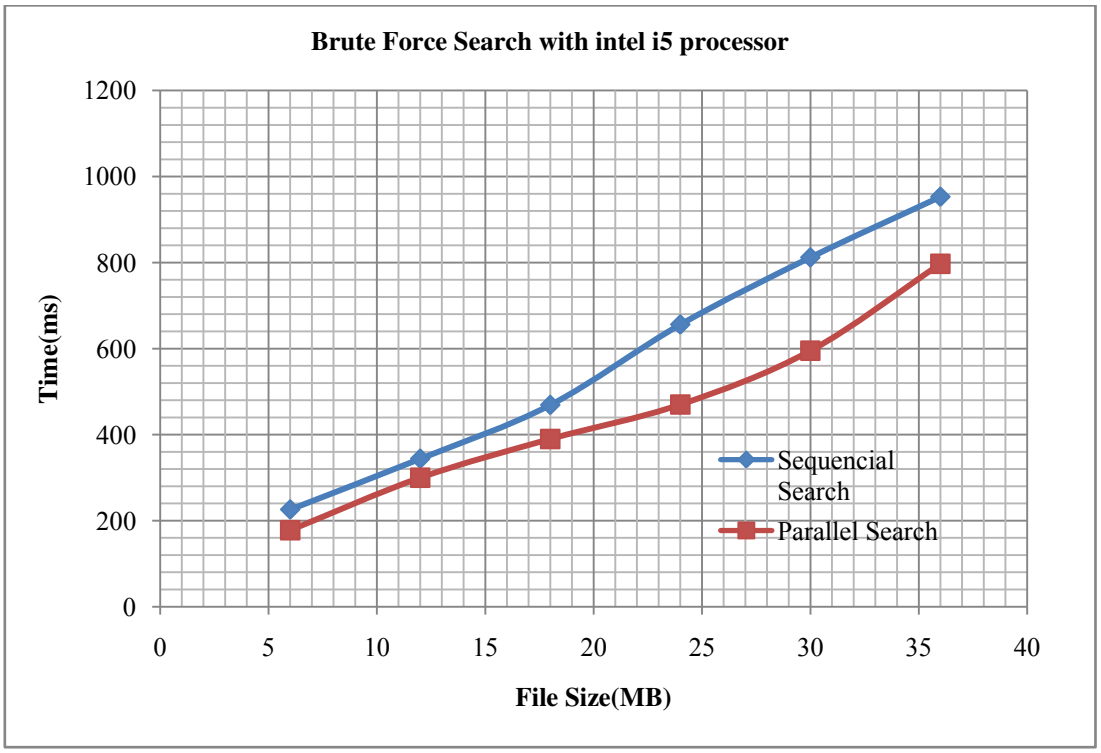

Figure 5 : Graph for Brute force (Time vs File size)

The Fig 6 shows(Graph) Execution time vs File size using KMP Algorithms by sequential search and parallel search with intel i5 processor. From the graph we clearly observe that parallel is much better than sequential search. The Fig 7 shows(Graph) Execution time vs Text length using Boyer Moore Algorithm by parallel search and sequential search with intel i5 processor. From the graph we clearly observe that parallel is much better than sequential search.

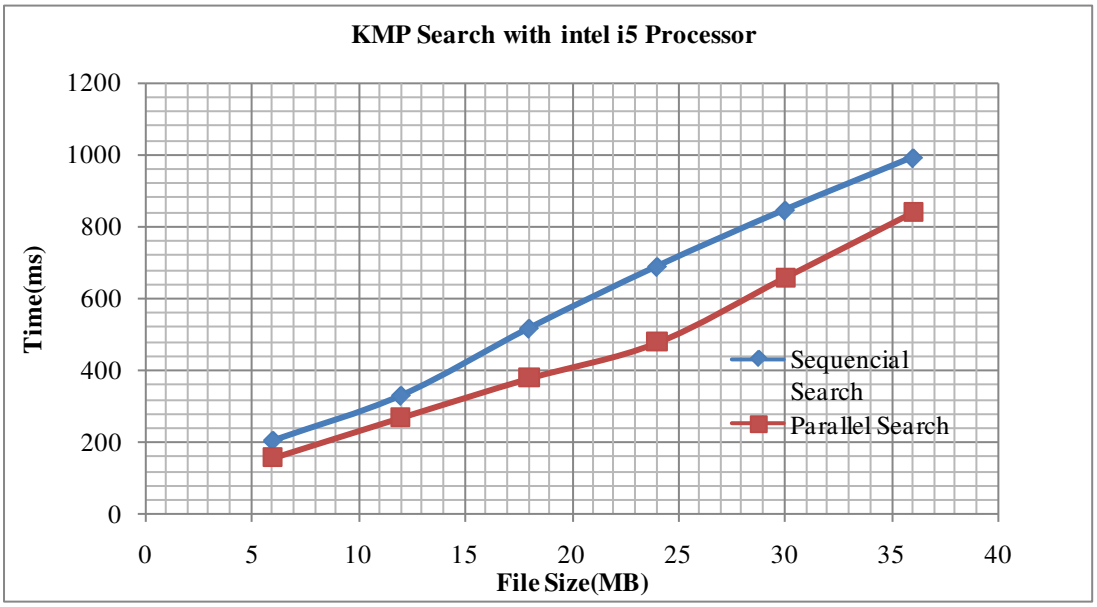

Figure 6 : Graph for KMP (Time vs File size)

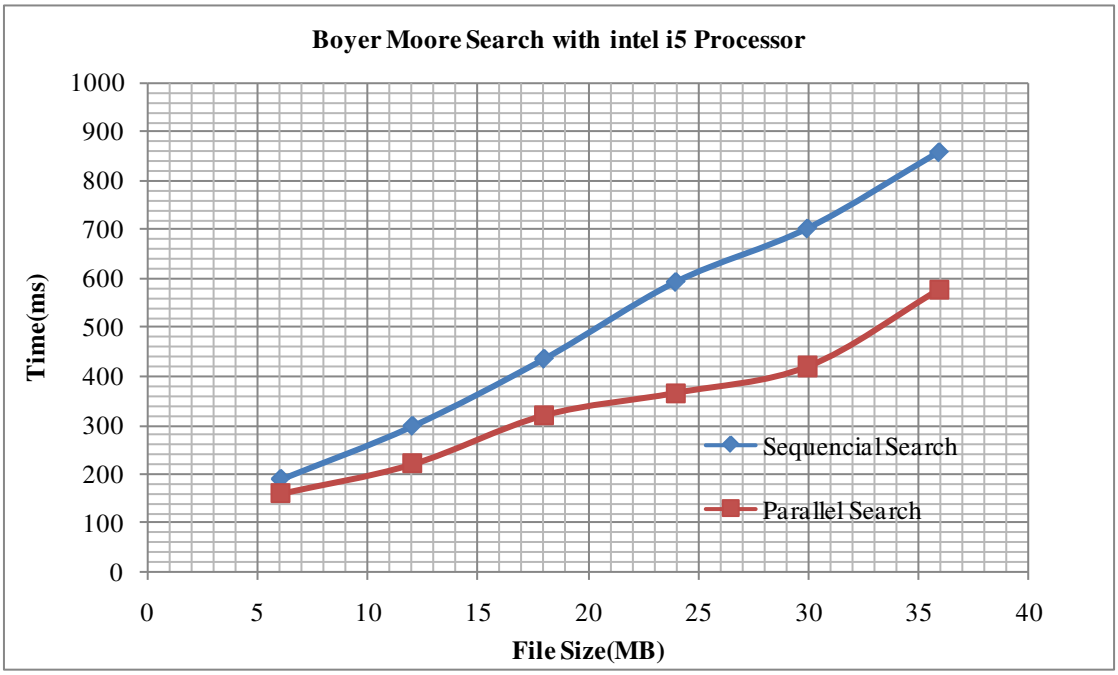

Figure 7 : Graph for Boyer Moore (Time vs Text Length)

The Fig 8 shows (Graph) Execution time vs File size on sequential search with intel i3 processor using Boyer Moore, Brute force, KMP Algorithm. This graph shows the performance difference between Boyer Moore, Knuth Morris Pratt and Brute force algorithms. From the graph clearly observe that BM is better compared to other approaches. The Fig 9 shows(Graph)Execution time vs File size on parallel search with intel i3 processor using Boyer Moore, Brute force, KMP Algorithm. This graph shows the performance difference between Boyer Moore, Knuth Morris Pratt and Brute force algorithms. From the graph clearly observe that BM is better compared to other approaches, as well as this parallel approach is much better compared to sequential approaches.

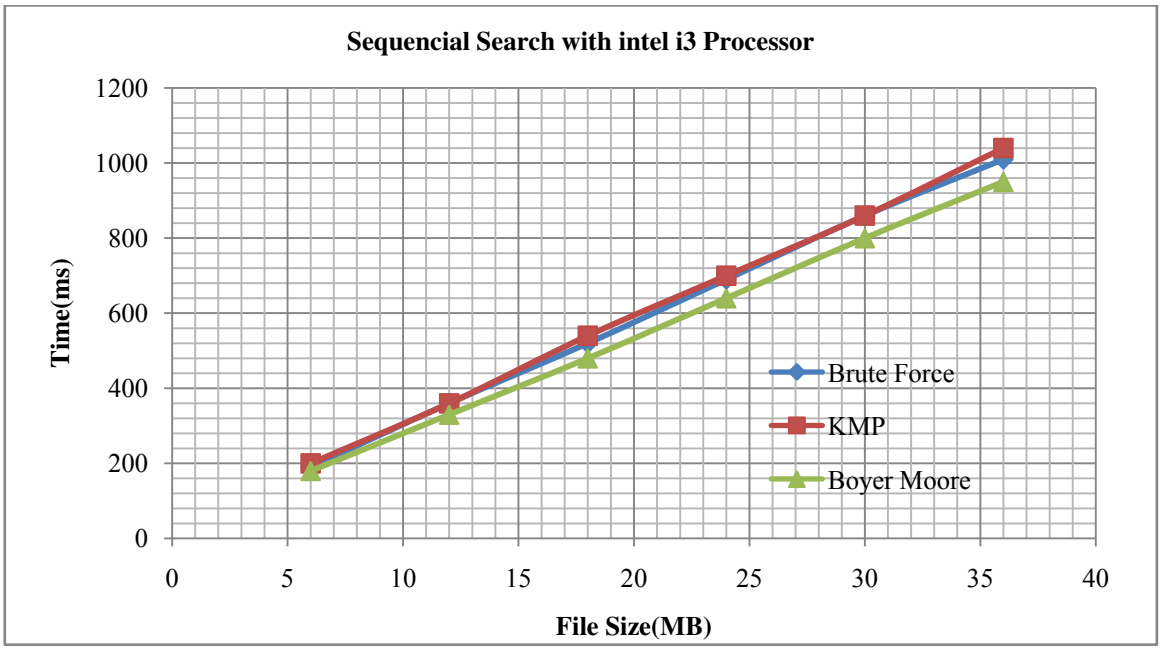

Figure 8 : Graph for sequential search (Time vs File size)

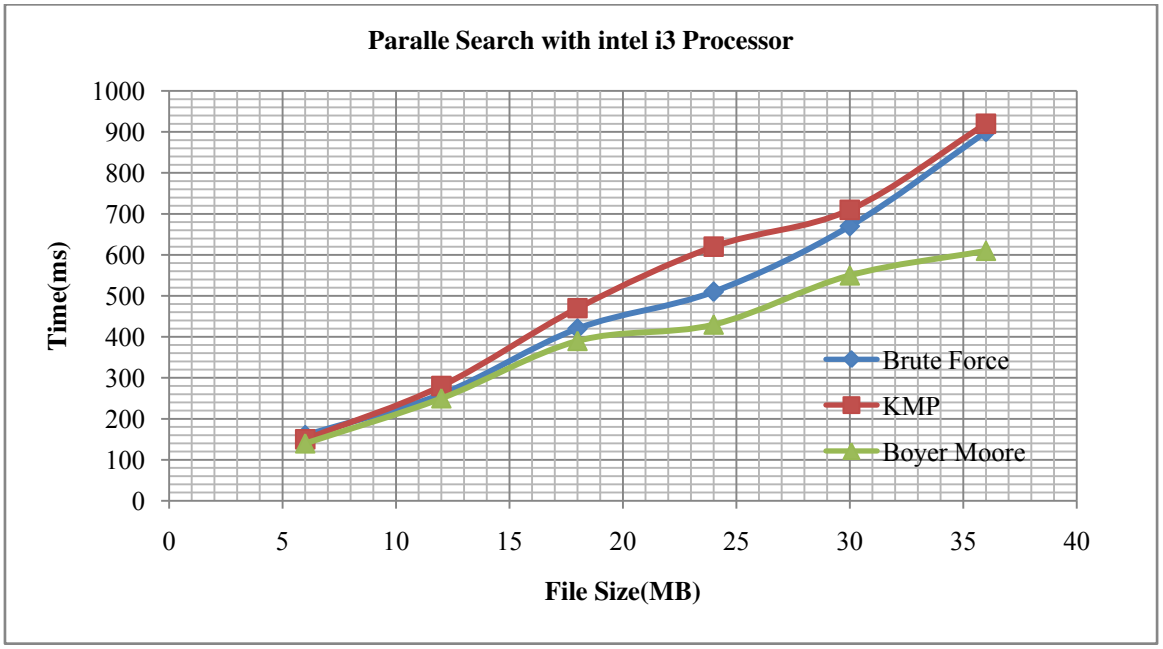

Figure 9 : Graph for parallel search (Time vs File size)

The Fig. 10 shows (Graph) Execution time vs file size of Brute force sequential search algorithm in Intel i3 and Intel i5 processor. From the graph we says that i5 is performed well compared to i3. The Fig 11 shows (Graph) shows Execution time vs File size using Brute force parallel search algorithm on intel i3 and i5 processor. From the graph we says that i5 is performed well compared to i3.

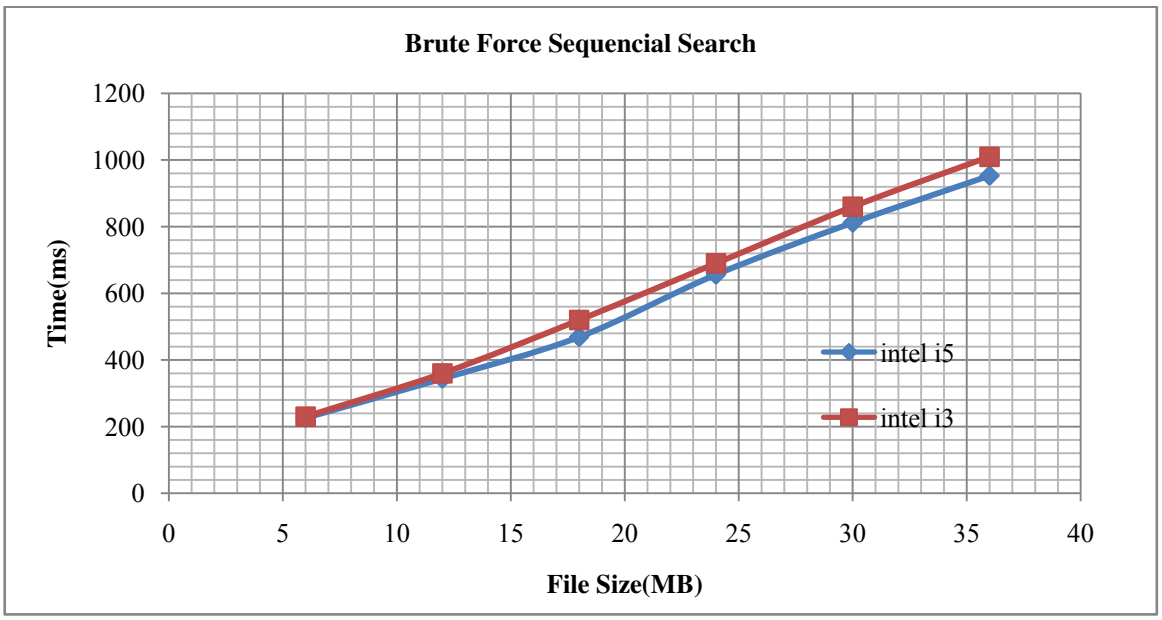

Figure 10 : Graph for Brute force(Time vs file size)

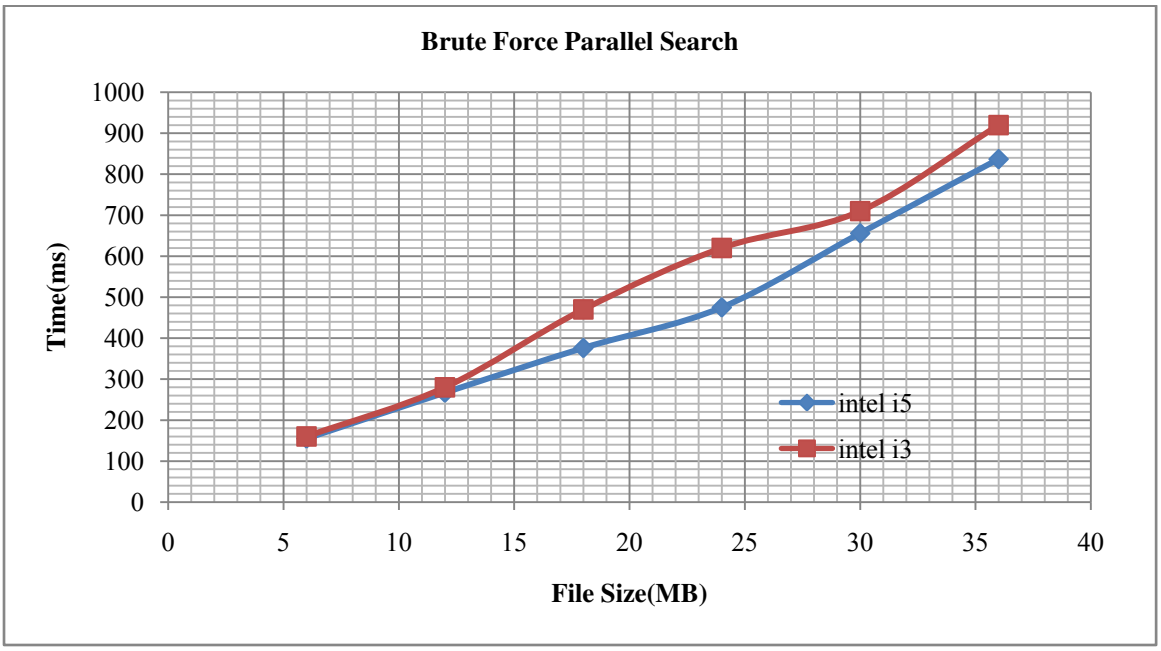

Figure 11 : Graph for Brute force parallel search (Time vs file size)

The Fig 12 shows (Graph) Execution time vs File size using KMP sequential search algorithm on Intel i3 and i5 processor. From the graph we says that i5 is performed well compared to i3. The Fig13 shows (Graph) Execution time vs File size using KMP parallel search algorithm on Intel i3 and i5 processor. From the graph we says that i5 is performed well compared to i3.

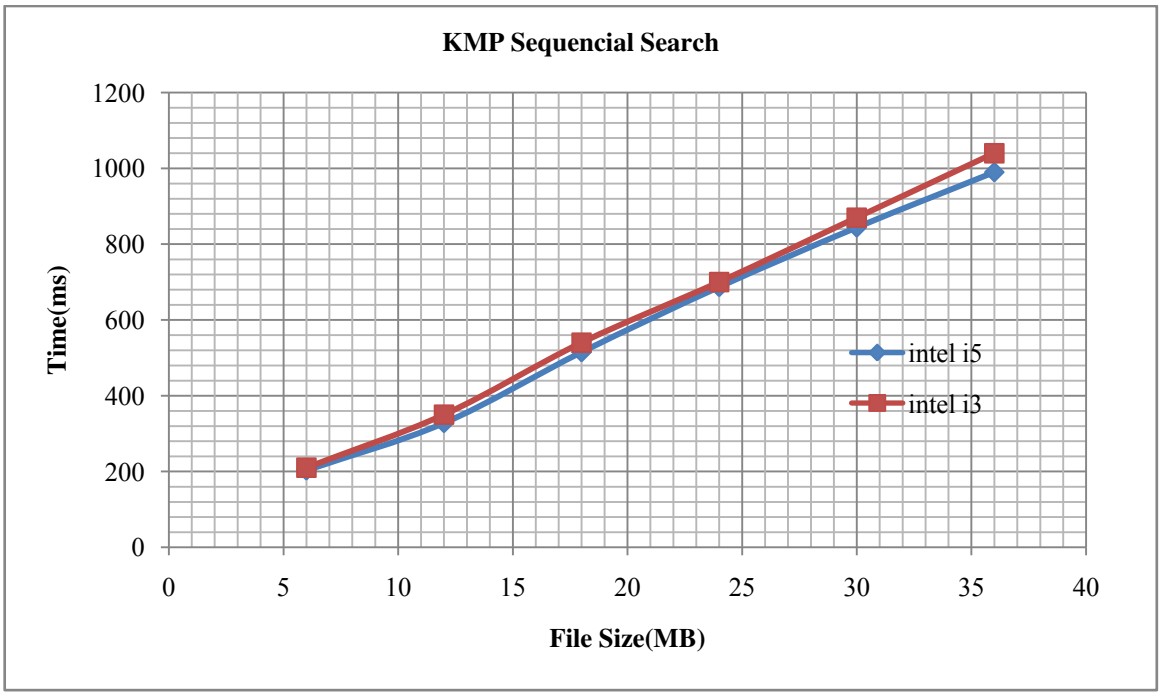

Figure 12 : Graph for KMP Sequential Search (Time vs file size)

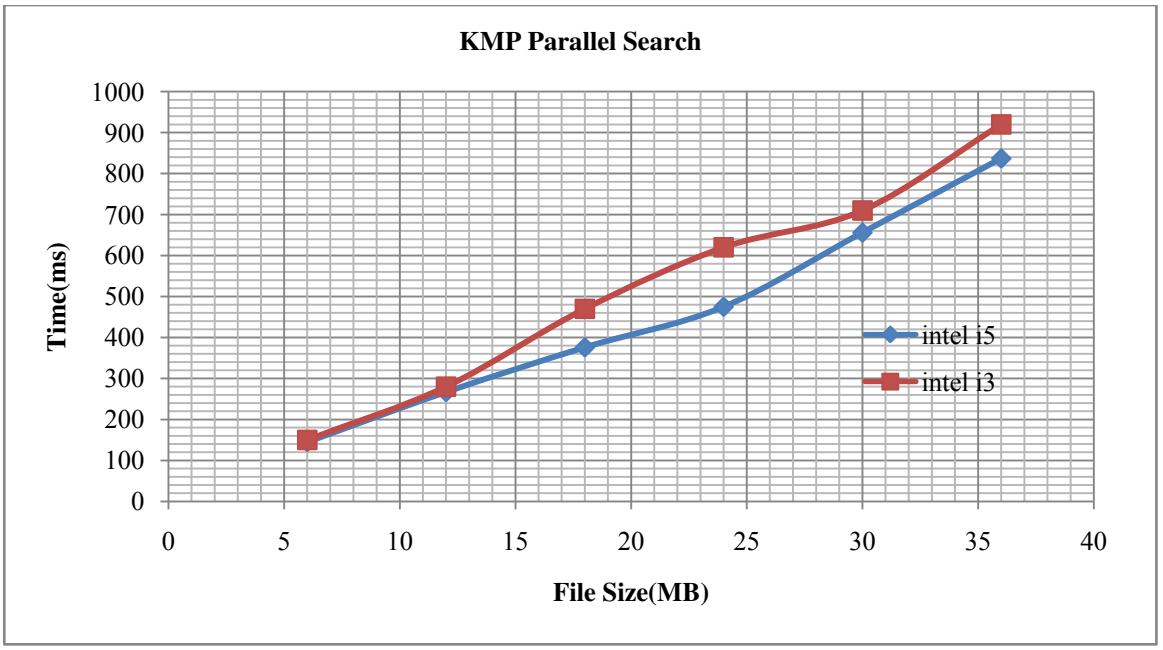

Figure 13 : Graph for KMP parallel Search (Time vs file size)

The Fig 14(Graph) shows Execution time vs File size using Boyer Moore sequential search algorithm on Intel i3 and i5 processor. From the graph we says that i5 is performed well compared to i3. The Fig 15(Graph) shows Execution time vs File size using Boyer Moore parallel search algorithm on Intel i3 and i5 processor. From the graph we says that i5 is performed well compared to i3.

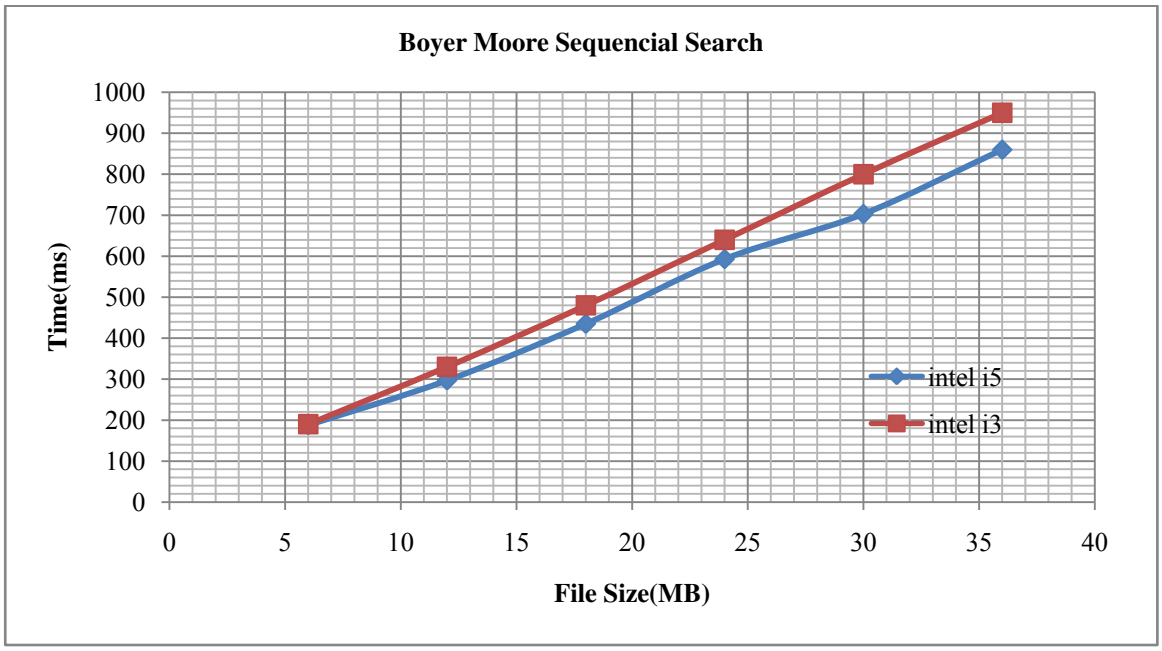

Figure 14 : Graph for Boyer moore sequential Search (Time vs file size)

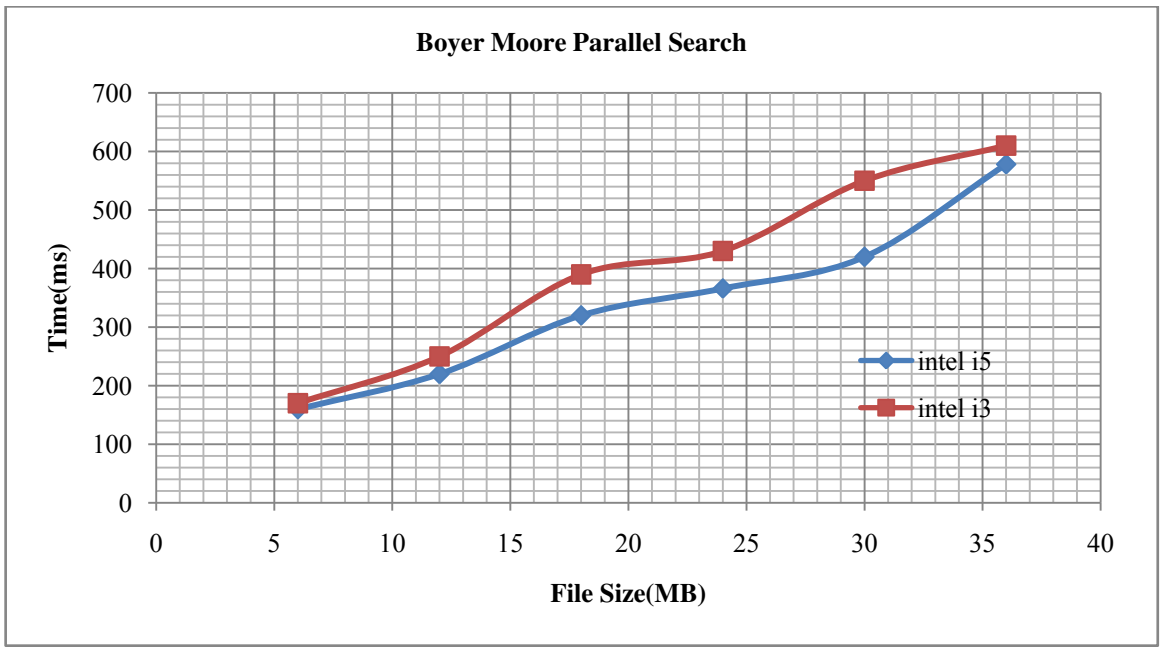

Figure 15 : Graph for Boyer Moore parallel Search (Time vs file size)

VI. Conclusions

In this paper we performed a comparative study on Knuth Morris Pratt, Boyer Moore and Brute force string matching algorithms based on the running time and in our tests with multicore processing, we used strings of varying lengths and texts of varying lengths. From the test results it is shown that the Boyer Moore algorithm is extremely efficient in most cases and Knuth-Morris-Pratt algorithm is not better on the average than the Brute force algorithm. We conclude that Boyer Moore string matching algorithm is the most efficient one among the three string matching algorithms with multicore processing compared to earlier versions. As a future enhancement, these algorithms can be compared with other efficient parallel string matching algorithms thereby finding the most efficient algorithm which can be used in many fields such as cryptography, molecular biology. Thus the problem of matching becomes easier.

References Références Referencias

1. Badoiu M. and Indyk P., "Fast Approximate Pattern Matching with Few Indels via Embeddings," ACM-

SIAM Symposium on Discrete Algorithms, pp. 651-652, 2004.
2. H. Kim, H. Hong, H.-S. Kim, and S. Kang, "A Memory-Efficient Parallel String Matching for Intrusion Detection Systems," IEEE Comm. Letters, vol. 13, no. 12, pp. 1004-1006, Dec. 2009.
3. Hyun Jin Kim, Hong-Sik Kim, Sungho Kang, "A Memory-Efficient Bit-Split Parallel String Matching Using Pattern Dividing for Intrusion Detection Systems," IEEE Transactions on Parallel and Distributed Systems, vol. 22, no. 11, pp. 1904-1911, Nov. 2011.
4. Mhashi M., Rawashdeh A., and Hammouri A.,"A Fast Approximate String Searching Algorithm," Computer Journal of Science Publication, pp. 405-412, 2005.
5. Dany Breslauer , Leszek Gąsieniec , Roberto Grossi, "Constant-Time word-size string matching", Proceedings of the 23rd Annual conference on Combinatorial Pattern Matching, Helsinki, Finland, 2012.
6. S. Faro and T. Lecroq, "The Exact Online String Matching Problem: a Review of the Most Recent Results", ACM Computing Surveys 45(2), 2013.
7. Jonathan L., "Analysis of Fundamental Exact and Inexact Pattern Matching Algorithms," Technical Document, Stanford University, 2004.
8. Chinta Someswararao, K Butchiraju, S ViswanadhaRaju, "Recent Advancement is Parallel Algorithms for String matching on computing models - A survey and experimental results", LNCS, Springer, pp.270-278, ISBN: 978-3-642-29279-8, 2011.
9. Chinta Someswararao, K Butchiraju, S ViswanadhaRaju, "PDM data classification from STEP- an object oriented String matching approach", IEEE conference on Application of Information and Communication Technologies, pp.1-9, ISBN: 978-1-61284-831-0, 2011.
10. Chinta Someswararao, K Butchiraju, S ViswanadhaRaju, "Recent Advancement is Parallel Algorithms for String matching - A survey and experimental results", IJAC, Vol 4 issue 4, pp-91-97, 2012.
11. Simon Y. and Inayatullah M., "Improving Approximate Matching Capabilities for Meta Map Transfer Applications," Proceedings of Symposium on Principles and Practice of Programming in Java, pp.143-147, 2004.
12. Chinta Someswararao, K Butchiraju, S ViswanadhaRaju, "Parallel Algorithms for String Matching Problem based on Butterfly Model", pp.41-56, IJCST, Vol. 3, Issue 3, July – Sept, ISSN 2229-4333, 2012.
13. Chinta Someswararao, K Butchiraju, S ViswanadhaRaju, "Recent Advancement is String matching algorithms- A survey and experimental results", IJCIS, Vol 6 No 3, pp.56-61, 2013.
14. S. viswanadha raju, "parallel string matching algorithm using grid", "international journal of distributed and parallel systems" (ijdps) vol.3, no.3, 2012.
15. Hyunjin kim and sungho kang, "A pattern group partitioning for parallel string matching using a pattern grouping metric", ieee communications letters, vol. 14, no. 9, pp. 878-880, 2010.
16. Daniel luchaup, l "speculative parallel pattern matching", "ieee transactions on information forensics and security, vol. 6, no. 2, pp.438-451, 2011".
17. Hyunjinkim, member, ieee, "A memory-efficient bit-split parallel string matching using pattern dividing for intrusion detection systems", ieee transactions on parallel and distributed systems, vol. 22, no. 11, pp.1904-1911, 2011".
18. Charalampos S, "String matching on a multicore gpu using cuda", "parallel and distributed processing laboratory department of applied informatics, university of macedonia 156 egnatia str, p.o. box 1591, 54006 thessaloniki, greece".
19. Thierry lecroq, "Fast exact string matching algorithms", "litis, faculty des sciences et des techniques, university de rouen, 76821 mont-saint-avignon cedex, france", january 2007.
20. Derek pao, "string searching engine for virus scanning" ieee transactions on computers, vol. 60, no.11, pp1596-1609, 2011.
21. Hassan ghasemzadeh," structural action recognition in body sensor networks: distributed classification based on string matching"," ieee transactions on information technology in biomedicine, vol. 14, no. 2, pp.425-435, 2010".
22. HyunJin Kim and Seung-Woo Lee, "A Hardware-Based String Matching Using State Transition Compression for Deep Packet Inspection", "ETRI Journal, Volume 35, Number 1, pp-154-157, 2013"
23. Ali Peiravi and Mohammad Javed Rahimzadeh, "A novel scalable and storage-efficient architecture for high speed exact string matching", "etri journal, volume 31, number 5, pp.545-553,2009".
24. Geist, A., Beguelin, A., Dongarra, J., Jiang, W., Manchek, R. & V., S. "PVM: Parallel Virtual Machine", A Users Guide and Tutorial for Networked Parallel Computing, MIT Press. 1994.
25. Grama, A., Karypis, G., Kumar, V. & Gupta, A. "Introduction to Parallel Computing", Addison Wesley, 2003.
26. Leow, Y., Ng, C. &W.F.,W.. "Generating hardware from OpenMP programs", International Conference on Field Programmable Technology, pp. 73–80, 2006.
27. Marr, D. T., Binns, F., Hill, D. L., Hinton, G., Koufaty, D. A., Miller, J. A. & Upton, M.. "Hyper-Threading

Technology Architecture and Microarchitecture", Intel Technology Journal, pp. 4–15, 2002.
28. R. S. Boyer and J. S. Moore, A fast string searching algorithm, Carom. Association of computing Machinary, pp-262–272, 1977.
29. Donald Knuth; James H. Morris, Jr, Vaughan Pratt, "Fast pattern matching in strings". SIAM Journal on Computing, pp-323–350, 1977.
30. Z. Galil, "Optimal parallel algorithms for string matching," in Proc. 16th Annu. ACM symposium on Theory of computing, pp. 240-248, 1984.

This page is intentionally left blank