IJCSC International Journal of Computer Science & Communication Vol. 1, No. 2, July-December 2010, pp. 271-275

String Searching Algorithm Implementation-Performance Study with Two Cluster Configuration

Prasad J.C[1] & K.S.M.Panicker[2]

[1]Dr.M.G.R.University Chennai cum Asst.Professor, CSE Dept. FISAT, Cochin, Kerala, India
[2]Centre for High Performance Computing, FISAT, Cochin, Kerala, India
Email: [1]cheeran.prasad@gmail.com, [2]mcpanicker@email.com

ABSTRACT

This paper consists of implementation of the eleven algorithms in two different cluster architecture environments. Static pattern of string matching algorithms are widely used for searching. For comparisons of speed different algorithms are implemented with parallel programming technique of MPI Standard, for a large text file with a pattern of small length, performance is based on eleven algorithms which are KMP, BM, BMH, ZT, QS, BR, FS, SSABS, TVBS, ZTBMH and BRBMH. This work consists of implementation of the same algorithms in two different cluster architecture environments. This has improved the reliability of these algorithms in a cluster environment. The result of pattern searching showed the variation of performance with this customized cluster architecture for variable number of compute nodes with different pattern lengths. Nearly 80% of the searching code part works in parallel. The performance of string search algorithm is based on the network bandwidth and the selection of algorithms used. Depending on certain applications, different algorithm can be used. It is observed that the second cluster searching architecture provides better speed with different algorithms.

Keywords: String Search, Cluster Architecture, MPI Programming, Searching Algorithm.

1. INTRODUCTION

The importance of pattern searching with different cluster architecture is more relevant with the latest advancements in DNA sequencing, web search engines, database operations, signal processing, error detection, speech and pattern recognition areas require pattern searching problem to process Terabytes amount of data. Most of the researchers usually focus on achieving high throughput with expensive resources of software and hardware [1]. The cluster architectures considered here is based on Beowulf architectures[2] with open source technologies with 1 Gbps speed of networks. During the benchmark test of HP Linpack system, our system worked with a speed of 80 Gigaflops and 96 Gigaflops with 32 nodes[17]. Therefore, this research paper focuses on providing a high-speed but low-cost string matching implementation details with limited resources in two different configurations. The implementation is based on Message Passing Implementation (MPI) standard based parallel programming technology.

2. STRING MATCHING ALGORITHMS

In string matching algorithm, the problem include parameters *m, n* and *O* where *m* is the length of the searched pattern, *n* is the length of the text being searched and *o* is the size of the alphabet. The Knuth-Morris-Pratt Algorithm[3] (KMP) was developed by D.Knuth J.Morris and V.Pratt in 1974. It compares the pattern with the text from left to right. In case of a mismatch or whole match it uses the notion border of the string. It decreases the time of searching compared to the Brute Force algorithm. The time complexity of the preprocessing phase is O(*m*) and of the searching phase is O(*nm*).

The Boyer-Moore algorithm(BM)[4] was developed by R.S.Boyer and J.C.Moore in 1977. It compares the pattern with the text from right to left. In case of a mismatch or whole match it uses two shift functions; the bad character shift and the good suffix shift. It is not very efficient for small alphabets. The time complexity of the preprocessing phase is O(m+ o) and of the searching phase is O(mn).

The Boyer Moore Horspool Algorithm[5,6](BMH) was developed by N.Horspool in 1980. It is a simplification of the Boyer-Moore algorithm. It compares the right most character in the pattern moving to the right. In case of a mismatch or whole match it uses only the Horspool bad character shift function with the right most character in the current text window. It is faster than, and easier to implement than the Boyer –Moore algorithm. The time complexity of the preprocessing phase is O(*m*+ *o*) and of the searching phase is O(*mn*).

The Zhu Takaoka algorithm[7] was developed by R.F.Zhu and T.Takaoka in 1989. It uses two consecutive text characters to compute the bad-character shift. In case of a mismatch or whole match it uses the preprocessing

hashing function. It is effective for multiple patterns matching in two-dimensional string matching. The time complexity of the preprocessing phase is $O(m+ o^2)$ and searching phase in $O(mn)$ time complexity.

The Quick Search Algorithm [8] was developed by Sunday in 1990. A quick search algorithm of string matching compares characters from the end of the search string to its beginning. When a character doesn't match, the next character in the text beyond the search string determines where the next possible match begins. It is fast for short patterns with large number of alphabets. Preprocessing phase is $O(m+ o)$ time and searching phase is $O(mn)$ time complexity.

Berry and Ravindran designed an algorithm(BR Algorithm[9]) in 1999, which performs the shifts by considering the bad-character shift for the two consecutive text characters immediately to the right of the window. It is a hybrid of the Quick Search and Zhu and Takaoka algorithms. Preprocessing phase is $O(m+ o 2)$ and searching phase in $O(mn)$ time complexity.

The Fast Searching algorithm[10] was developed by D. Cantone and S. Faro in 2003. It is a variation of Boyer-Moore Algorithm. It compares the pattern with the text from right to left. In case of a mismatch or whole match it uses the bad character function only if the mismatching character is the last character of the pattern, otherwise the good-suffix function is to be used. It is efficient in very short pattern.

The SSABS algorithm[11] was developed by S.S.Sheik, S.K.Aggarwal, A.Poddar, N.Balakrishnan and K Sekar in 2004. It scans the pattern with the text in new order to attain maximum efficiency as following. It compares the right most character in the pattern with the right most character in the text window, then it start from the left most character in the pattern with the left most character in the text window then it will scan the next last character of the pattern with the next last character of the text window and then it goes backward to the left. In case of a mismatch or whole match it uses the Quick Search bad-character shift value.

The TVSBS algorithm [12] was developed by R.Thathoo, A.Virmani, S.S.Lakshmi, N.Balakrishnan and K.Sekar in 2006. It is a combination of Berry Ravindran and SSBAS algorithms. It scans the pattern with the text using searching phase as the SSABS algorithm. In case of a mismatch or whole match it uses the BR bad-character shift value. Preprocessing phase time complexity of the proposed algorithm is O(o + k * o) and space complexity is O(o + k * o). The time complexity is $O([n/(m + 2)])$ in the best case and $O(m(n – m + 1))$ in the worst case.

The ZTMBH algorithm[13] was developed by Y.Huang, X.Pan, Y.Gao and G.Cai in 2008. It is a combination of Zhu Takaoka and Boyer Moore Horsepool algorithms. It scans the pattern with the text using search phase as the BMH algorithm. In case of a mismatch or match it uses the BR bad-character shift value.

The BRBMH algorithm[14] was developed by Ahmad Fadel Klaib and Hugh Osborne in Dec 2008. BRBMH offers improved number of comparison and elapsed searching time when compared to the well known algorithm such as BF, KMP, BM, BMH, KR, ZT, QS, BR, FS, SSABS, TVBS, ZTBMH and BRFS algorithms.

3. MASTER WORKER MODEL STATIC PATTERN SEARCHING ENVIRONMENT

A single string pattern matching problem can be defined as follows. Let T be a large text of n number of character size and P be a pattern of length m. Text $T = T[1]\ T[2]$ ……..$T[n]$ and Pattern $P = P[1]\ P[2]$…….$P[m]$. The characters of both T and P belong to a finite set of elements of the set S and $m<<n$. Search processes identify all the occurrence of the pattern P in text T. We have considered two types of input data (natural language input string and DNA sequence string) for the evaluation of algorithms. The actual task of searching is done parallel among the processors from 0 to $p – 1$. Master node decompose the text in to r subtexts and distributed to available workers(nodes). Each subtext contains $k = [(n-m+1)/r] + m – 1$ characters, where k is the successive characters of the complete text. There is an overlap of m-1 successive characters between successive sub texts. So there will be a redundancy of $r(m – 1)$ characters for processing. Our objective is to compare the result of searching with different algorithms. So redundancy of searching does not have relevance in our system.

4. PARALLEL SEARCHING ALGORITHM

Knuth-Morris-Pratt Algorithm (KMP),Boyer-Moore algorithm(BM), Boyer Moore Horspool Algorithm, The Zhu Takaoka algorithm, quick search algorithm, BR Algorithm, Fast Searching algorithm, SSABS algorithm,TVSBS algorithm and ZTMBH algorithm are implemented with our cluster environment. In order to present the algorithm, we make the following assumptions.

(i) The processors have an identifier and are numbered from 0 to p – 1.

(ii) The files are stored in a common memory area and it can be accessed by all the processors.

From the master and the worker sub-procedures [16], it is clear that the application ends only when all the local string matching operations have been completed and their results have been collected by the master processor.

```
Main procedure
main ()
{
  1. Initialize message passing routines;
  2. If (process_id==master) then //Master node job
  {
2.1. Decompose the text in to r subtexts with length k= [ (n-m+1) / r ] + m-1
  2.2 Broadcast the string pattern, concerned subtexts to work nodes;
        (MPI_Send)
  2.3. Receive the results (i.e. number of occurrences of each subtext) from all workers
  2.4. Find the sum of number of occurrences from each workers;
  2.5. Print the total number of occurrences of P in files (i.e.total matches).
  }
  3. else //Work nodes job
  {
  3.1. Receive the pattern and subtext (MPI_Receive)
  3.2. Call matches=search (filename) ; //using any static string matching algorithm.
  3.3. Send the results (i.e. no: of occurances) to the master;
  }
  4. Exit message passing operations;
}
```

MPI is a library that allows you to do problems in parallel using message passing to communicate between processes. After including "<mpi.h>" header file, MPI codes must start with MPI_Init before doing any MPI work. Likewise, they should all issue a MPI_Finalize when they are done. Besides these most basic of MPI routines, MPI_Comm_Rank routine is used to determine what the number of the Processing Element(PE) the routine is running on is. This will always be from 0 to n-1 for n PEs. The same code is running on each of the PEs. Unless you want the same codes to use the same data in exactly the same manner and generate exactly the same results on each node (which is kind of pointless), you will want to have the PEs vary their behaviour based upon their PE number. In general, though, the PE number will be used to load different data files or take different branches in the code. In most cases, the processes would need to communicate. This is done by passing messages which use the routines MPI_SEND and MPI_RECIEVE.

Search part will be changed for different algorithm. Depending on the length of the pattern to be searched and the number of alphabet set in the character set, number of processors, the result varies for different algorithms. Hardware architecture of this cluster consists of two servers - Network Information Service (NIS) server and Job Scheduling Server. Configurations are same for both: IBM X series Quad Core Xeon processors, 146 GB SAS HDD, 2 GB RAM, NIC 2G. Computational Nodes configurations: Intel Pentium 4 with HT, 3.0 Ghz, 256 MB RAM, 80 GB HDD with S.M.A.R.T.

5. RESULTS OF SEARCHING

A comparative study on the performance of today's most prominent retrieval models is presented as described above. The result compares the results of searching with sequential and two Beowulf cluster configuration (*Dakshina I* & *Dakshina II*) implementations to identify the parameters involved in searching. To get a reliable and consistent performance result, the average of ten executions for different pattern of constant length is given in the table as result values.

Table 1
Sequential Execution Time in Second for a File Size 12 MB

m	*KMP*	*BM*	*BMH*	*ZT*	*QS*	*BR*	*FS*	*SSABS*	*TVSBS*	*ZTMBH*	*BRB MH*
5	1.470	0.783	0.786	1.296	0.523	0.673	0.523	0.796	0.667	0.963	0.482
10	1.414	0.754	0.746	1.243	0.480	0.664	0.512	0.723	0.654	0.921	0.456
15	1.401	0.703	0.712	1.156	0.452	0.631	0.481	0.694	0.642	0.809	0.423
20	1.381	0.644	0.664	1.074	0.435	0.602	0.463	0.662	0.605	0.653	0.397
25	1.372	0.536	0.576	1.007	0.413	0.574	0.433	0.614	0.576	0.462	0.368

Table 2
Parallel Execution Time in Seconds for a File Size 12 MB with 5 Nodes in Beowulf Cluster Dakshina-1

m	*KMP*	*BM*	*BMH*	*ZT*	*QS*	*BR*	*FS*	*SSABS*	*TVSBS*	*ZTMBH*	*BRBMH*
5	0.531	0.284	0.284	0.469	0.189	0.244	0.189	0.287	0.241	0.348	0.174
10	0.510	0.272	0.269	0.449	0.175	0.240	0.183	0.261	0.236	0.322	0.166
15	0.475	0.254	0.257	0.417	0.163	0.228	0.174	0.251	0.232	0.292	0.154
20	0.464	0.232	0.240	0.388	0.158	0.217	0.167	0.239	0.218	0.236	0.144
25	0.450	0.194	0.208	0.364	0.150	0.208	0.156	0.222	0.208	0.167	0.134

Table 3
Parallel Execution Time in Second for a File Size 12 MB with 10 Nodes Beowulf Cluster Dakshina-I

m	*KMP*	*BM*	*BMH*	*ZT*	*QS*	*BR*	*FS*	*SSABS*	*TVSBS*	*ZTMBH*	*BRBMH*
5	0.411	0.219	0.221	0.398	0.146	0.189	0.146	0.223	0.187	0.270	0.136
10	0.395	0.212	0.210	0.369	0.135	0.187	0.143	0.203	0.183	0.258	0.129
15	0.376	0.199	0.199	0.351	0.128	0.178	0.134	0.194	0.182	0.228	0.119
20	0.368	0.180	0.187	0.345	0.123	0.169	0.131	0.187	0.169	0.183	0.110
25	0.361	0.150	0.167	0.334	0.117	0.160	0.122	0.172	0.162	0.129	0.104

Table 4
Parallel Execution Time in Seconds for a File Size 12 MB with 5 Nodes in Beowulf Cluster Configuration-I1

m	*KMP*	*BM*	*BMH*	*ZT*	*QS*	*BR*	*FS*	*SSABS*	*TVSBS*	*ZTMBH*	*BRBMH*
5	0.304	0.161	0.162	0.292	0.106	0.138	0.106	0.161	0.137	0.196	0.098
10	0.289	0.156	0.153	0.268	0.097	0.135	0.103	0.149	0.134	0.189	0.093
15	0.278	0.144	0.145	0.254	0.093	0.131	0.097	0.142	0.132	0.165	0.086
20	0.264	0.132	0.138	0.253	0.090	0.124	0.096	0.136	0.122	0.132	0.081
25	0.261	0.107	0.122	0.247	0.084	0.118	0.088	0.124	0.117	0.093	0.075

Table 5
Parallel Execution Time in Second for a File Size 12 MB with 10 Nodes Beowulf Cluster Configuration-II

m	*KMP*	*BM*	*BMH*	*ZT*	*QS*	*BR*	*FS*	*SSABS*	*TVSBS*	*ZTMBH*	*BRBMH*
5	0.106	0.055	0.053	0.096	0.034	0.046	0.036	0.053	0.047	0.064	0.034
10	0.096	0.051	0.050	0.086	0.032	0.044	0.035	0.052	0.046	0.063	0.030
15	0.093	0.041	0.045	0.084	0.031	0.041	0.032	0.048	0.045	0.054	0.028
20	0.088	0.039	0.047	0.083	0.030	0.038	0.031	0.046	0.042	0.043	0.027
25	0.079	0.035	0.039	0.081	0.028	0.037	0.028	0.042	0.038	0.030	0.025

6. CONCLUSION

Parallel implentation of KMP, BM, BMH, ZT, QS, BR, FS, SSABS, TVSBS, ZTMBH, BRBMH algorithms were tested. The speedup of a program using multiple processors in parallel computing is limited by the time needed for the sequential fraction of the program as per Amdajl's law[15]. Around 80 % code part of our searching programmecan be run parallel. In the case of parallelization, Amdahl's law states that if P is the proportion of a program that can be made parallel (i.e. benefit from parallelization), and (1 " P) is the proportion that cannot be parallelized (remains serial), then the maximum speedup that can be achieved by using N processors is $S = \frac{1}{(1-P)+P/N}$. Thus theoretical speedup with various algorithms is S= $\frac{5N}{N+4}$. The result proves the order of searching for various algorithms. Speed up factor do not exactly matches with the result. It is because of the potential bottlenecks such as network communication overheads, memory bandwidth and I/ O bandwidth[20]. Since size of our problem is fixed, Gustafson's Law is not applicable with our search results. Future work of this experiment is to predict the results based on the pattern length and string length.

REFERENCES

[1] P. D. Michalidis and K. G. Margaritis, "On-Line Approximate String Searching Algorithms: Survey and Experimental Results", *Intern. J. Computer Math.*, 2002, **79(8)**, pp. 867–888.

[2] Thomas Sterling, *Beowulf Cluster Computing with Linux*, MIT Press, Oct 2001, pp. 77– 79.

[3] D.E.Knuth, J.H.Morris and V.R.Pratt, "Fast Pattern Matching in Strings", *SIAM Journal on Computing*, **6**, No.2, pp.323-350, 1977.

[4] R. S. Boyer and J. S. Moore, "A Fast String Searching Algorithm," *Communications of the ACM*, **20**, Session10, Oct.1977, pp.761– 772.

[5] T.RAITA, "Tuning the Boyer-Moore-Horspool String Searching Algorithm", *Software Practice and Experience*, **22**, pp 879-884, 1992.

[6] R.N.Horspool, "Practical Fast Searching in Strings", *Software-Practice and Experience*, **10**, No.6, pp.501-506, 1980.

[7] R.F.Zhu, T.Takaoka, "On Improving the Average Case of the Boyer-Moore String Matching Algorithm", *Journal of Information Processing*, **10**, No.3, pp.173-177, 1987.

[8] Sunday, "A Very Fast Substring Search Algorithm", *Common ACM*, No.33, 1990. pp.132-140.

[9] Berry and Ravindran, "Fast String Matching Algorithm and Experimental Results", *Proceedings of the Prague Stringology Club*, pp.16-26, 2001.

[10] D.Cantone, S.Faro, "Fast-search: A New Efficient Variant of the Boyer-Moore String Matching Algorithm", *Lecture Notes in Computer Science*, **2647**, pp.47-58, 2003.

[11] S.S.Sheik, S.K.Aggarwal, A.Poddar, N.Balakrishnan and K.Sekar, "*A Fast Pattern Matching*, 2004. pp. 1251-1256.

[12] R.Thathoo,A.Virmani,S.S.Lakshmi, N. Balakrishnan and K.Sekar, "TVSBS: A Fast Exact Pattern Matching Algorithm for Biological Sequences", *Current Science*, **91**, No.1, pp.47-53, 2006.

[13] Y.Huang, L.Ping, X.Pan and G. Cai, "A Fast Pattern Matching Algorithm for Biological Sequences", *IEEE*, pp.608-611, 2008.

[14] Ahmad Fadel Klaib, Hugh Osborne, "Searching Protien Sequence Databases Using BRBMH Matching Algorithm", *IJCSNS International Journalof Computer Science and Network Security*, **8**, No 12, Dec 2008.

[15] Amdahl, Gene. "Validity of the Single Processor Approach to Achieving Large-Scale Computing Capabilities", *AFIPS Conference Proceedings,* **30** : 483–485. 1967.

[16] Panagiotis D. Michailidis, Konstantinos G. Margaritis, *Parallel Implementations for String Matching Problem on a Cluster of Distributed Workstations*, *Neural, Parallel & Scientific Computations*, **10**, Issue 3, Year of Publication: 2002, pp: 287–312.

[17] Prasad J.C., K.S.M.Panicker, "Beowulf-Dhakshina Cluster Architecture with Linux Debian Operating System for MPI Programming", *Proceedings of International Conference on Information Processing(ICIP 2009)*, Aug 7-9, 2009, ISBN: 978-93-80026-75-2, Bangalore, India. Pp: 350-355.