Global Journal of Computer Science and Technology
Hardware & Computation
Volume 13 Issue 2 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

A Frame work for Parallel string Matching- A Computational Approach with Omega Model

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

Abstract- Now a day's parallel string matching problem is attracted by so many researchers because of the importance in information retrieval systems. While it is very easily stated and many of the simple algorithms perform very well in practice, numerous works have been published on the subject and research is still very active. In this paper we propose a omega parallel computing model for parallel string matching. Experimental results show that, on a multi-processor system, the omega model implementation of the proposed parallel string matching algorithm can reduce string matching time by more than 40%.

Keywords: string matching; parallel string matching; computing model; omega model.

GJCST-A Classification : C.1.4

Strictly as per the compliance and regulations of:

A Frame work for Parallel String Matching- A Computational Approach with Omega Model

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

Abstract- **Now a day's parallel string matching problem is attracted by so many researchers because of the importance in information retrieval systems. While it is very easily stated and many of the simple algorithms perform very well in practice, numerous works have been published on the subject and research is still very active. In this paper we propose a omega parallel computing model for parallel string matching. Experimental results show that, on a multi-processor system, the omega model implementation of the proposed parallel string matching algorithm can reduce string matching time by more than 40%.**

Keywords: string matching; parallel string matching; computing model; omega model.

I. Introduction

String matching has been one of the most extensively studied problems in computer engineering since it performs important tasks in many applications like information retrieval (IRS), web search engines, error correction and several other fields [1-12]. Especially with the introduction of search engines dealing with tremendous amount of textual information presented on the World Wide Web, so this problem deserves special attention and any improvements to speed up the process will benefit these important applications [1-12].

As current free textual databases are growing almost exponentially with the time, the string matching problem becomes impractical to use the fastest sequential algorithms on a conventional sequential computer system [1-12]. To improve the performance of searching on large text collections, some researchers developed special purpose algorithms called parallel algorithms that parallelized the entire database comparison on general purpose parallel computers where each processor performs a number of comparisons independently. In Parallel processing the text string T and pattern P are assumed and that two input words have already been allocated in the processors in such a way that each processor stores a single text symbol, and some processors additionally a single pattern symbol. The input words are stored symbol-by-symbol in consecutive processors numbered according to the snake - like row - major indexing, that is, the processors in the odd-numbered rows 1, 3, 5, ... are numbered from left to right, and in the even-numbered rows from right to left. (The first symbols of T and P are in processor 1, the next in processor 2, and so on.) This allocation scheme places symbols adjacent in the text or pattern in adjacent processors. The output of the string matching algorithm is that each processor is to be marked as either being a starting position of an occurrence of P in T or not. In this paper we proposed a parallel string matching technique based on butterfly model.

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 butterfly computing models. This work suggests the most efficient algorithmic butterfly 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. Recent Advancements and Global Research

The first optimal parallel string matching algorithm was proposed by Galil [13]. On SIMD-CRCW model, this algorithm required n / log n processors, and the time complexity is O(log n) ; on SIMD-CREW model, it required n / log2 n processors and the time complexity is O(log2 n). Vishkin [14] improved this algorithm to ensure it is still optimal when the alphabet size is not fixed. In [15], an algorithm used O (n × 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 [16]. 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.

SV Raju et.al [17] presents new method for exact string matching algorithm based on layered architecture and two-dimensional array. This has applications such as string databases and computational biology. The main use of this method is to reduce the time spent on comparisons of string matching by distributing the data among processors which achieves a linear speedup and requires layered architecture and additionally p*# processors.

Author α : Associate Professor, Department of CSE, GRIET, Hyderabad, AP, India.
Author σ : Assistant Professor, Dept of CSE, SRKR Engineering College, Bhimavaram, AP, India.
Author ρ : Professor & HOD, Department of CSE, JNTU College of Engineering, JNTU Jagithyal, AP, India.

Bi Kun et.al [18] proposed the improved distributed string matching algorithm. And also an improved single string matching algorithm based on a variant Boyer-Moore algorithm is presented. In this they implement algorithm on the above architecture and the experiments prove that it is really practical and efficient on distributed memory machine. Its computation complexity is $O(n/p + m)$, where n is the length of the text, and m is the length of the pattern, and p is the number of the processors. They show that this distributed architecture is suitable for paralleling the multipattern string matching algorithms and approximate string matching algorithms.

Hsi-Chieh Le [19] et.al presents three algorithms for string matching on reconfigurable mesh architectures. Given a text T of length n and a pattern P of length m, the first algorithm finds the exact matching between T and P in O(1) time on a 2-dimensional RMESH of size (n - m+ 1) x m. The second algorithm finds the approximate matching between T and P in O(k) time on a 2D RMESH, where k is the maximum edit distance between T and P. The third algorithm allows only the replacement operation in the calculation of the edit distance and finds an approximate matching between T and P in constant-time on a 3D RMESH. By this paper we state that this is simpler model would be sufficient to run the proposed algorithms without increasing the reported time complexities.

S V Raju [20] et.al considers the problem of string matching algorithm based on a two-dimensional mesh. This has applications such as string databases, cellular automata and computational biology. The main use of this method is to reduce the time spent on comparisons in string matching by using mesh connected network which achieves a constant time for mismatch a text string and. This is the first known optimal-time algorithm for pattern matching on meshes. The proposed strategy uses the knowledge from the given algorithm and mesh structure.

Its'hak Dinstein [21] et.al propose a parallel computation approach to two dimensional shape recognition. This approach uses parallel techniques for contour extraction, parallel computation of normalized contour-based feature strings independent of scale and orientation, and parallel string matching algorithms. The implementation on the EREW PRAM architecture is discussed, but it can be adapted to other parallel architectures.

Jin Hwan Park [22] et.al presents efficient dataflow schemes for parallel string matching. In this they consider two sub problems known as the exact matching and the k-mismatches problems are covered. Three parallel algorithms based on multiple input (and output) streams are presented. Time complexities of these parallel algorithms are O((n/d)+a), 0 £ a £ m, where n and m represent lengths of reference and pattern strings (n >> m) and d represents the number of streams used (the degree of parallelism). They show, they can control the degree of parallelism by using variable number (d) of input (and output) streams. They show their approaches solve the exact matching and the k-mismatches problems with time complexities of O((n / d) + a), where a = log m for the hierarchical scheme, m for the linear scheme, and 0 for the broadcasting scheme. Required time to process length n reference string is reduced by a factor of d by using d identical computation parts in parallel. With linear systolic array architecture, m PEs are needed for serial design and d*m PEs are needed for parallel design, where m is the pattern size and the d is the controllable degree of the parallelism (i.e. number of streams used).

S V Raju [23] et.al considers the problem of exact string matching algorithm based on a two-dimensional array. This has applications such as string databases, cellular automata and computational biology. The main use of this method is to reduce the time spent on comparisons in string matching by finding common characters in pattern string which achieves a constant time O(1) for pattern string in a text string. This reduces many calls across backend interface.

Chuanpeng Chen [24] et.al propose a high throughput configurable string matching architecture based on Aho-Corasick algorithm. The architecture can be realized by random-access memory (RAM) and basic logic elements instead of designing new dedicated chips. The bit-split technique is used to reduce the RAM size, and the byte-parallel technique is used to boost the throughput of the architecture. By the particular design and comprehensive experiments with 100MHz RAM chips, one piece of the architecture can achieve a throughput of up to 1.6Gbps by 2-byte-parallel input, and we can further boost the throughput by using multiple parallel architectures.

Prasanna[25] et.al propose a multi-core architecture on FPGA to address these challenges. They adopt the popular Aho-Corasick (AC-opt) algorithm for our string matching engine. Utilizing the data access feature in this algorithm, they design a specialized BRAM buffer for the cores to exploit a data reuse existing in such applications. Several design optimizations techniques are utilized to realize a simple design with high clock rate for the string matching engine. An implementation of a 2-core system with one shared BRAM buffer on a Virtex-5 LX155 achieves up to 3.2 GBPS throughput on a 64 MB state transition table stored in DRAM. Performance of systems with more cores is also evaluated for this architecture, and a throughput of over 5.5 Gbps can be obtained for some application scenarios.

S. Muthukrishnan et.al[26] present an algorithm on the CRCW PRAM that checks if there exists a false match in O(1) time using O(n) processors. This algorithm does not require preprocessing the pattern. Therefore, checking for false matches is provably

simpler than string matching since string matching takes O(log(log m)) time on the CRCW PRAM. In this hey use this simple algorithm to convert the Karp–Rabin Monte Carlo type string-matching algorithm into a Las Vegas type algorithm without asymptotic loss in complexity. Finally they present an efficient algorithm for identifying all the false matches and, as a consequence, show that string-matching algorithms take A.log log m/ time even given the flexibility to output a few false matches.

S V Raju [27] et.al present new approach for parallel string matching. Some known parallel string matching algorithms are considered based on duels by witness who focuses on the strengths and weaknesses of the currently known methods. The new 'divide and conquer' approach has been introduced for parallel string matching, called the W-period, which is used for parallel preprocessing of the pattern and has optimal implementations in a various models of computation. The idea, common for every parallel string matching algorithm is slightly different from sequential ones as Knuth-Morris-Pratt or Boyer-Moore algorithm.

III. Communication Network Relation to Computer System Components

A typical distributed system is shown in Figure 1. Each computer has a memory-processing unit and the computers are connected by a communication network. Figure 2 shows the relationships of the software components that run on each of the computers and use the local operating system and network protocol stack for functioning.

The distributed software is also termed as middleware. A distributed execution is the execution of processes across the distributed system to collaboratively achieve a common goal. An execution is also sometimes termed a computation or a run. The distributed system uses a layered architecture to break down the complexity of system design. The middleware is the distributed software that drives the distributed system, while providing transparency of heterogeneity at the platform level [28]. Figure 2 schematically shows the interaction of this software with these system components at each processor.

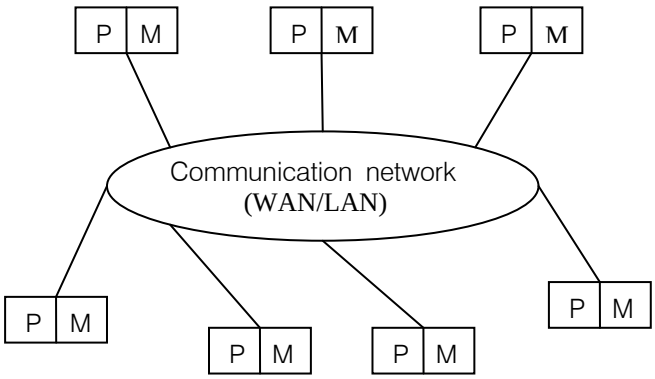

Figure 1 : A distributed systems connects processors by a communication network.

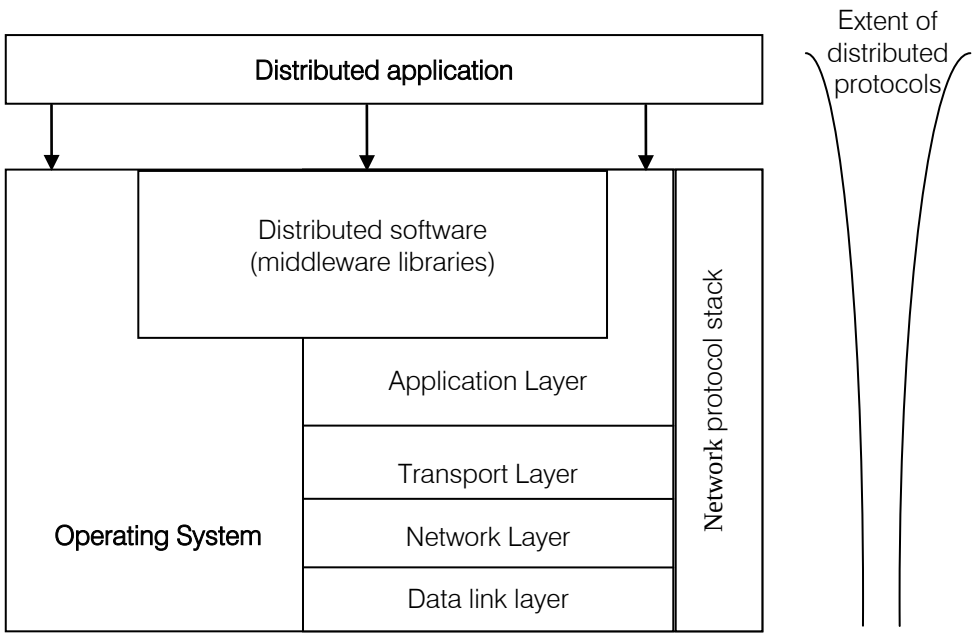

Figure 2 : Interaction of the software components at each processor.

IV. Text Partitioning

The exact string-matching problem can achieve data parallelism with data partitioning technique. We decompose the text into r subtexts, where each subtext contains (T/p)+m-1 successive characters of the complete text. There is an overlap of m-1 string characters between successive subtexts, i.e, a redundancy of r(m-1) characters. Alternatively it could be assumed that the database of an information retrieval system contains r independent documents. Therefore, in both the cases all the above partitions yield a number of independent tasks each comprising some data (i.e. a string and a large subtext) and a sequential string matching procedure that operates on that data. Further, each task completes its string matching operation on its local data and returns the number of occurrences[29-31]. Finally, we can observe that there are no communication requirements among the tasks but only global (or collective) communication is required.

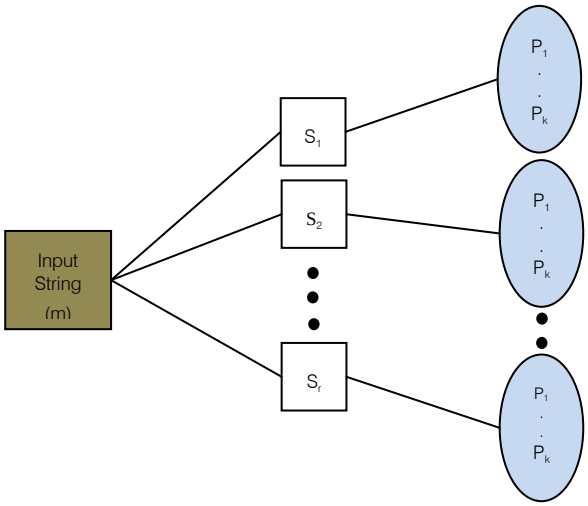

Figure 3 : Framework for pool of Processors

The main issue to be addressed is how the several tasks (or r subtexts) can be mapped or distributed to multiple processors for concurrent execution. In [29-31] different ways of distributing the database across a multi computer network were discussed. Let p be the number of processors in network and r be the number of subtext in the whole collection then the text partition is defined as, if r=p then each subtext contains T/p+m-1 characters. This is called static allocation of subtext as shown in Fig 3. In the next section we present the parallel algorithm that is based on static allocation of subtext using MPI library. A significant contribution of this paper is a demonstration of the maximum size buffer with 2k processors for implementation of string matching and capable of accepting a character from r subtexts where k=8bits. This architecture enables a buffered string matching system implementing a KMP like pre computation algorithm. In the above mapping {a1, a2…ar} is the input string where r represents subtext and {p1, p2…pk} are the number of processors for the given input string (k=8). In the above mapping the given input string will be allocated to each processor as shown in Fig. 3.

V. Methodology

Multistage interconnection networks (MINs) consist of more than one stages of small interconnection elements called switching elements and links interconnecting them. Multistage interconnection networks (MINs) are used in multiprocessing systems to provide cost-effective, high-bandwidth communication between processors and/or memory modules. A MIN normally connects N inputs to N outputs and is referred as an N × N MIN. The parameter N is called the size of the network[32-33]. The popularity of MINs stems from both the operational features they deliver –e.g. their ability to route multiple communication tasks concurrently- and the appealing cost/performance ratio they achieve. MINs with the Banyan [34] property e.g. Omega Networks [35], Delta Networks [36], and Generalized Cube Networks [37] are more widely adopted, since non-Banyan MINs have -generally- higher cost and complexity. Both in the context of parallel and distributed system, the performance of the communication network interconnecting the system elements (nodes, processors, memory modules etc) is recognized as a critical factor for overall system performance. Consequently, the need for communication infrastructure performance prediction and evaluation has arisen, and numerous research efforts have targeted this area, employing either analytical models (mainly based on Markov models and Petri-nets) or simulation techniques.

There are several different multistage interconnection networks proposed and studied in the literature. Figure1 illustrates a structure of multistage interconnection network, which are representatives of a general class of networks. This Figure 4 shows the connection between p inputs and b outputs, and connection between these is via n number of stages.

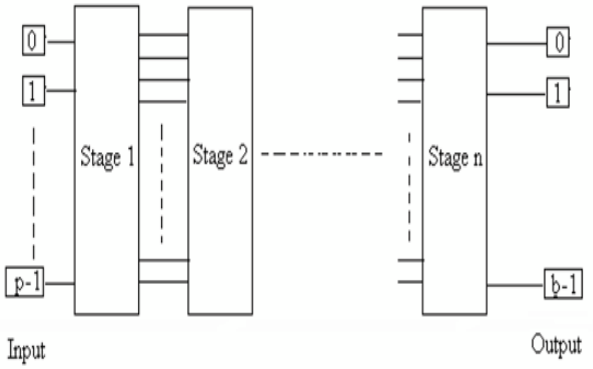

Figure 4 : A Multistage Interconnection Network(MIN)

A multistage interconnection network is actually a compromise between crossbar and shared bus networks multistage interconnection networks are:

- Attempt to reduce cost
- Attempt to decrease the path length

In a multistage interconnection network, as in a crossbar, switching elements are distinct from processors. Instead messages pass through a series of switch stages. The network can be constructed from unidirectional or bi-directional switches and links. In a unidirectional MIN, all messages must traverse the same number of wires, and so the cost of sending a message is independent of processor location. In effect, all processors are equidistant. In a bidirectional MIN, the number of wires traversed depends to some extent on processor location, although to a lesser extent than a mesh or hypercube.

VI. Omega Network

Omega network connecting P processors to P memory banks as shown in Fig 5 In general, it consists of $p = (q + 1)2^q$ processors, organized as q + 1 ranks of 2^q processors each (Figure 1). Optionally we shall identify the rightmost and the leftmost ranks, so there is no rank q, and the processors on ranks 0 and q - 1 are connected directly. Let us denote the processor i on the rank r by $P_{ir,r}$, $0 \le i < 2^q$, $0 \le r \le q$. Then processor $P_{i,r+1}$ is connected to the two processors $P_{i,r}$, and $P_{ir,r}$ and processor $P_{ir,r+1}$ is connected to the two processors $P_{i,r}$. and $P_{ir,r}$. Recall that $i^r = i_{q-1} .. .. i_r..i_0$ These four connections form a "butterfly" pattern, from which the name of the network is derived. The hypercube is actually the butterfly with the rows collapsed. The communication link in the hypercube between processors P_i, and P_{ir} is identified with the communication links in the butterfly between $P_{i,r+l}$ and $P_{ir,r+1}$, and between $P_{ir.r+l}$ and $P_{i.r}$.

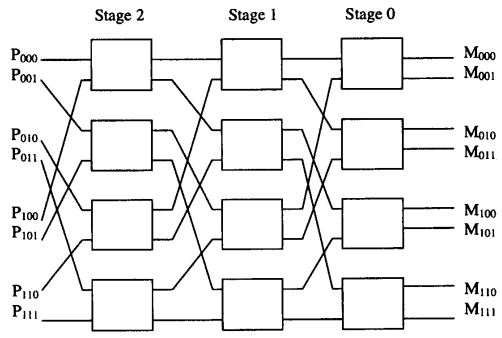

Figure 5 : Omega Network

Terminology : Terminal reliability is defined as the probability of successful communication between an input output pair. In this section, terminal reliability of Omega, has evaluated. The Omega is a unique-path MIN that has N input switches and N output switches and n stages, where n = log2N. An 8×8 Omega has three stages, 12 SEs and 32 links. And reliability is shown in Fig 6. Let r be the probability of a switch being operational. As Omega is a unique- path MIN, the failure of any switch will cause system failure, so from the reliability point of view, there are log2N SEs in series for each terminal path. Hence, the terminal reliability of an N×N Omega is R_t (Omega) = $(r)^{\log_2 N}$ As there is only a single path between a particular input Si , i =1, 2, 3, 4, and a output in an 8×8 Omega so the terminal reliability is R_t (Omega) = $(r)^3$.

Switching Reliability	Terminal Reliability of Omega
0.99	0.970299
0.98	0.941192
0.96	0.884736
0.95	0.857375
0.94	0.830584
0.92	0.778688
0.9	0.729

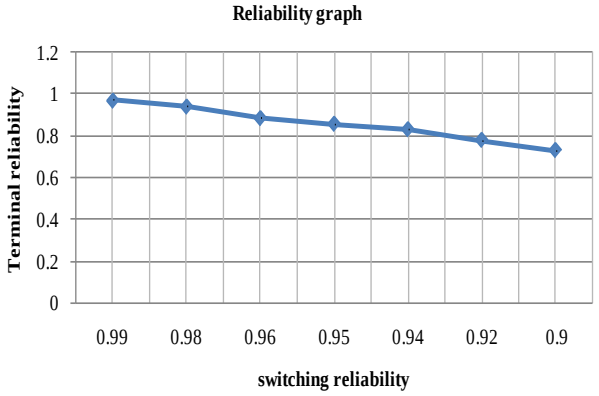

Figure 6 : Switching Reliability

VII. Proposed System Structure

In this we propose a system for parallel processing with omega model. Its shared-memory, expandable MIMD parallel computer.

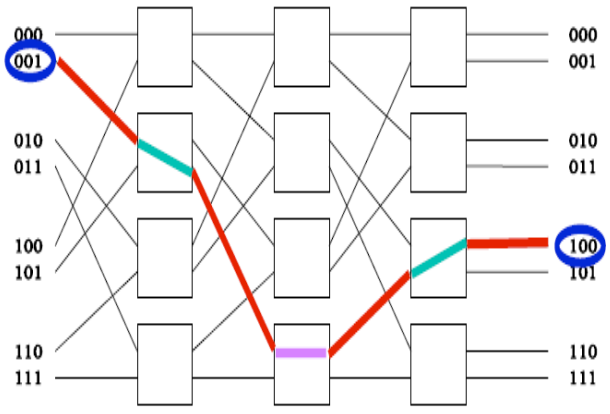

Figure 7 : Omega Network with 8 i/o

The computer got its name from the omega switch which it uses for interprocessor communication.

The switch supports a processor-to-processor bandwidth of 32 Mbits/second. Figure 7 illustrates 8-input 8-output omega switch.

VIII. Programming the Omega Parallel Processor

The omega Parallel Processor is programmed exclusively in high-level programming languages. Searching, IRS, Editing, compiling and linking, downloading, running and debugging of programs are done from a UNIX front-end. A window manager enables rapid switching between the front-end and the Butterfly system environments. Two distinct approaches to programming the omega have seen widespread use: message passing and shared memory. When using the message passing paradigm the programmer decomposes the application into a moderately sized collection of loosely coupled processes which from time to time exchange control signals or data. This approach is similar to programming a multiprocessor application for a uniprocessor. In the shared memory approach, a task is usually some small procedure to be applied to a subset of the shared memory. A task, therefore, can be represented simply as an index, or a range of indices, into the shared memory and an operation to be performed on that memory. This style is particularly effective for applications containing a few frequently repeated tasks. Memory and processor management are used to keep all memories and processors equally busy.

IX. Parallel String Matching Algorithm on Omega Model

In this model data on processors have been organized such that they represent the m sets of length of n-m+1 of the text string with m* n-m+1 matrix plus, the first processor of each row segment holding the first element of each set also carries an element of pattern. The process is similar as per above for the remaining m 1 rows. First show how to find the occurrences of pattern P in text string T on omega model with m*(n m+1) in constant time O (1).

Algorithm for string matching (pattern P, text T)
Input: m elements of pattern initially distributed to the m processors on the first column, one processor and $L_{i,j}$ distributed to the m*(n-m+1) processors on the row and column. Where $1 \le i \le m$ and $m \le j \le m*(n-m+1)$
Starting processors passes elements on row buses: each first processor $p_{i,1}$, where $1 \le i \le m$, broadcasts the element ci,1 to every processor in the ith row using only row buses .After this multiple row broadcast communication operations each processor $P_{i,j}$ saves received element as $r_{i,j}$. **Compare and Set results:** Each processor $P_{i,j}$ compares $r_{i,j}$ with $X_{i,j}$, if $r_{i,j} = X_{i,j}$ if sets result=1 otherwise, result=0 **Sum up 1's:** perform a one-dimensional binary prefix sum operation on each column simultaneously for the value of result. Each processor $P_{i,j}$ where $1 \le (i,j) \le m$ stores the binary prefix sums to $b_{i,j}$. **Distance:** If $P_{m,j} = 0$ then the string matching(i.e. exact string matching) otherwise approximate string matching with k mismatches.

Lemma-1 : Each step in the above algorithm runs in constant time. Thus we have the following theorem.

Theorem 1.1 : There is a constant time string matching algorithm on a omega model that finds the occurrences of pattern in text using m*(n-m+1) processors

Example : Text string T(n)= GOKARAJU and Pattern string P(m)= RAJU L1 ={GOKAR}, L2={OKARA} L3={KARAJ}L4={ARAJU}

Step-1

R	G	O	K	A	R
A	O	K	A	R	A
J	K	A	R	A	J
U	A	R	A	J	U

Step-2

<R,G>	<R,O>	<R,K>	<R,A>	<R,R>
<A,O>	<A,K>	<A,A>	<A,R>	<A,A>
<J,K>	<J,A>	<J,R>	<J,A>	<J,J>
<U,A>	<U,R>	<U,A>	<U,J>	<U,U>

Step-3

0	0	0	0	1
0	0	1	0	1
0	0	0	0	1
0	0	0	0	1

Step-4

0	0	0	0	1
0	0	1	0	1
0	0	0	0	1
0	0	0	0	1
4	4	3	4	0

As per the given example, after step 4 in the matrix $M_{m+1,j}$ values useful for deciding the matching is exact string matching or approximate string matching with the k mismatches.

Lemma-2 : So that the string matching is completely scalability and obtain the following theorem.

Theorem 1.2 : The given two strings size of text n and size of pattern m. find the occurrences of pattern in text.

There is completely scalable on Butterfly model. The algorithm runs in O(m*(n-m+1))/P time, where P is the number of processors and $1 \le p \le m*(n-m+1)$.

X. Performance Evaluation

In order to evaluate the overall performance of a multi-priority (*N*x*N*) MIN consisting of (2x2) SEs, we use the following metrics. Let *T* be a relatively large time period divided into *u* discrete time intervals ($\tau 1, \tau 2, , \tau u$).

Average throughput the average number of packets accepted by all destinations per network cycle. Formally, Th_{avg} (or *bandwidth*) is defined as

$$Th_{avg} = \lim_{u \to \infty} \frac{\sum_{k=1}^{u} n(k)}{u} \qquad (1)$$

where *n(k)* denotes the number of packets that reach their destinations during the *kth* time interval.

Normalized throughput is the ratio of the *average throughput Thavg* to number of network outputs *N*. Formally, *Th* can be expressed by back ground and reflects how effectively network capacity is used.

$$Th = \frac{Th_{avg}}{N} \qquad (2)$$

Relative normalized throughput RTh(i) of i-class priority traffic, where $i=1..p$ is the *normalized throughput Th(i)* of *i*-class priority packets divided by the corresponding-class *offered load* $\lambda(i)$ of such packets.

$$RTh(i) = \frac{Th(i)}{\lambda(i)} \qquad (3)$$

Average packet delay Davg(i) of i-class priority traffic, where $i=1..p$ is the average time a corresponding-class priority packet spends to pass through the network. Formally, D*avg(i)* is expressed by

$$D_{avg}(i) = \lim_{u \to \infty} \frac{\sum_{k=1}^{n(u)} t_d(k)}{n(u)} \qquad (4)$$

where *n(u)* denotes the total number of the corresponding- class priority packets accepted within *u* time intervals and *td(k)* represents the total delay for the *k*th such packet. We consider *td(k) = tw(k) + ttr(k)* where *tw(k)* denotes the total queuing delay for *kth* packet waiting at each stage for the availability of a corresponding-class empty buffer at the next stage queue of the network.

The second term *ttr(k)* denotes the total transmission delay for *kth* such packet at each stage of the network, that is just *n*nc*, where $n=\log_2 N$ is the number of intermediate stages and *nc* is the network cycle.

XI. Conclusions

In this paper we concentrate on parallel algorithms for string matching on computing models, especially in omega model. In this paper simulate the parallel algorithms for the implementation of high speed string matching; this uses fine-grained parallelism and performs matching of a search string by splitting the string into a set of substrings and then matching all of the substrings simultaneously. We also see that this implementation can be optimized in terms of resource utilization.

References Références Referencias

1. 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.
2. 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.
3. 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.
4. 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.
5. 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.
6. 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.
7. Chinta Someswararao, " Parallel String Matching Problems with Computing Models - An Analysis of the Most Recent Studies", International Journal of Computer Applications , Vol.76(15),pp.7-25,

Published by Foundation of Computer Science, New York, USA, 2013.
8. Chinta Someswararao, "Parallel String Matching with Multi Core Processors-A Comparative Study for Gene Sequences", Global Journal of Computer Science and Technology, Vol-13, Issue-1, pp.27-41, 2013.
9. K, Grabowski S, "Average-Optimal String Matching", Journal of Discrete Algorithms, pp- 579-594,2009.
10. Luis Russo L, Navarro G, Oliveira A, Morales P, "Approximate String Matching with Compressed Indexes Algorithm", pp- 1105-1136,2009.
11. Ilie L, Navarro G, Tinta L, "The Longest Common Extension Problem, Revisited and Applications to Approximate String Searching", Journal of Discrete Algorithms, pp-418-428, 2010.
12. Fredriksson K, Grabowski S, "Average-Optimal String Matching, Journal of Discrete Algorithms", pp- 579-594,2009.
13. Z. Galil, "Optimal parallel algorithms for string matching," in Proc. 16th Annu. ACM symposium on Theory of computing, pp. 240-248, 1984.
14. U. Vishkin, "Optimal parallel matching in strings," Information and control, vol. 67, pp. 91-113, 1985.
15. Y. Takefuji, T. Tanaka, and K. C. Lee, "A parallel string search algorithm", IEEE Trans. Systems, Man and Cybernetics, vol. 22, pp. 332-336, March-April 1992.
16. 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.
17. Viswanadha Raju, S.; Vinaya Babu, A.; Mrudula, M.; "Backend Engine for Parallel String Matching Using Boolean Matrix", IEEE on PAR ELEC, pp-281-283,2006.
18. Bi Kun, Gu Nai-jie, Tu Kun, Liu Xiao-hu, and Liu Gang A Practical Distributed String Matching Algorithm Architecture and Implementation World Academy of Science, Engineering and Technology , 2005.
19. Hsi-Chieh Leet,Fikret Ercalt, "RMESH Algorithms For Parallel String Matching" ,IEEE,1997.
20. S. Viswanadha Raju and A. Vinayababu, "Optimal Parallel algorithm for String Matching on Mesh Network Structure", 2006.
21. Its'hak Dinstein, ad M. Landau, "Using Parallel String Matching Algorithms for Contour Based 2-D Shape Recognition", IEEE,1990.
22. Jin Hwan Park and K. M. George, "Parallel String Matching Algorithms Based on Dataflow", IEEE on System Sciences, 1999.
23. S.Viswanadha Raju S R Mantena A.Vinaya Babu G V S Raju, "Efficient Parallel Pattern Matching using Partition Method",2006.
24. Chuanpeng Chen, Zhongping Qin, "A Bit-split Byte-parallel String Matching Architecture",IEEE,2009.
25. Qingbo Wang, Viktor K. Prasanna, "Multi-Core Architecture on FPGA for Large Dictionary String Matching", IEEE on Field Programmable Custom Computing Machines,2009.
26. S. Muthukrishnan "Detecting False Matches in String-Matching Algorithms", Algorithmica ,Springer-Verlag New York Inc.1997.
27. S.Viswanadha Raju , A.Vinaya Babu, G.V.S.Raju, and K.R. Madhavi , "W-Period Technique for Parallel String Matching",2007.
28. Ajay D. Kshemkalyani and Mukesh Singhal," Distributed Computing: Principles, Algorithms, and Systems",Cambridge.
29. S.Viswanadha Raju and A.Vinayababu, 2004, "Performance in the design of Parallel Programming", Proc ObComAPC-2004, Allied Publications, pp.380 to 392.
30. S.Viswanadha Raju, A.Vinayababu, S.P.Yanaiah and GVSRaju, 2006 "Parallel Approach for K String Matching", Proc NCIMDiL-2006, Indian Institute Of Technology, Kharagpur , 5-10.
31. J. Garofalakis, and E. Stergiou "An analytical performance model for multistage interconnection networks with blocking", Procs. of CNSR 2008,May 2008
32. Josep Torrellas, Zheng Zhang. The Performance of the Cedar Multistage Switching Network. IEEE Transactions on Parallel and Distributed Systems, 8(4), pp. 321-336, 1997.
33. Bhogavilli S. K., Abu-Amara H., "Design and Analysis of High-performance Multistage Interconnection Networks", IEEE Transactions on Computers, vol. 46, no. 1, January 1997, pp. 110 - 117.
34. G. F. Goke, G.J. Lipovski. "Banyan Networks for Partitioning Multiprocessor Systems" Procs. of 1st Annual Symposium on Computer Architecture, pp. 21-28, 1973.
35. D. A. Lawrie. "Access and alignment of data in an array processor", IEEE Transactions on Computers, C-24(12):11451155,Dec. 1975.
36. J.H. Patel. "Processor-memory interconnections for mutliprocessors", Procs. of 6th Annual Symposium on Computer Architecture. New York, pp. 168-177, 1979.
37. G. B. Adams and H. J. Siegel, "The extra stage cube: A fault-tolerant interconnection network for supersystems", IEEE Trans. on Computers, 31(4)5, pp. 443-454, May 1982.