

Optimal Parallel algorithm for String Matching on Mesh Network Structure

S. Viswanadha Raju and A. Vinayababu

*Dept of Computer Science and Engg. GRIET, Hyderabad 500072,
Andhra Pradesh, India.*

*Dept of Computer Science and Engg. JNTUniversity, Hyderabad 500072,
Andhra Pradesh, India*

E-mails: viswanadha_raju2004@yahoo.co.in, dravinayababu@yahoo.com

Abstract

In this paper we consider 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 we obtained $O(\sqrt{K})$ -time algorithm on $\sqrt{K} \times \sqrt{K}$ meshes ($K \leq n/2$) from $O(\sqrt{n})$ -time algorithm on $\sqrt{n} \times \sqrt{n}$ meshes for string matching and also reduced half of the processors. This is the first known optimal-time algorithm for pattern matching on meshes.

Keywords: Arrays, Boolean matrix, String matching, mesh structure.

1. Introduction

String matching mainly deals with the problem of finding all the occurrences of a string in a given text. String matching is one of the most extensive problems in computer technologies during past two decades. Pattern matching is widely implemented in information retrieval, web search engine, DNA sequencing, artificial intelligence and several other fields[4].

String matching problem can be defined as: consider a two strings one text string $T=T[1]T[2]T[3]...T[n]$ of length n and the other one pattern string(set of characters C-256) $P=P[1]P[2]P[3]....P[m]$ of length m (C-256) where both text and pattern string(or C-256) are sequences of characters from \hat{c} with $n \leq m$. Sequential algorithms for this string matching problem can be found in [1,4,5,7]. The parallel

algorithm solving for the string matching problem on a mesh structure. A mesh is two dimension grid in which there is processor at each grid point. It is closely related model is the linear array. A linear array consists of processors connected as follows: processor i is connected to the processors $i-1$ and $i+1$ for $2 \leq i \leq p-1$ processors 1 and p bounded processors. This parallel computer architecture is a $\sqrt{n} \times \sqrt{n}$ array of n processors interconnected according to the grid pattern.

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 the main result is improve the optimal from $O(\sqrt{n})$ -time algorithm for $\sqrt{n} \times \sqrt{n}$ meshes[9] to $O(\sqrt{K})$ -time algorithm for $\sqrt{K} \times \sqrt{K}$ meshes($K \leq n/2$) with the help of mesh connected array of processors and also half the processors are reduced. The structure of an algorithm is based on the data partition technique [18] which is most useful to preprocessed string. This is the first known optimal algorithm for pattern matching on meshes.

Most of recent research concerning meshes has concentrated on such communication problems as routing or sorting. There is however a growing body of results concerning other algorithmic problems like multisearch problems [2], and computational-geometry problems [1]. This paper contributes to this line of research, and, by presenting an optimal algorithm for the classical problem of pattern matching on mesh structure.

2. Linear Array of Processors

In this section text algorithms on array of processors are described. These algorithms are used as functions in the pattern-matching algorithm on a two-dimensional mesh. An array of n processors is built from processors $P_0 \dots P_{n-1}$, where P_i , is connected to P_{i-1} and P_{i+1} , if they exist. The text is assumed to have been already allocated in the processors, such that the processor P_i stores the i^{th} symbol of text $S = T_0 \dots T_{n-1}$. The pattern $P_0 \dots P_{m-1}$ is an input to the array, the symbols are fed into P_{n-1} and then transmitted through the array. The order of the input symbols is P_{m-1} through P_0 ,

In the next subsection an algorithm to search a text for a pattern on a array of processors is presented. This algorithm, called ARRAY PATTERN MATCHING (APM in short), is often used in what follows[13]. Three other algorithms for an array of processors are developed: one finds the period of a pattern, the other the witnesses of a pattern and Boolean matrix of text and pattern (or character set $C-256$) based on the first two algorithms. All of them are modifications of APM. All of these algorithms together are referred to as APM.

Finding Occurrences of a Pattern

The transmission of information through the line of processors is described in terms of packet routing, each of which carries a pattern(S -256) symbol and some attached useful messages and generated boolean matrix/ sparse table [18], which is used for static text string T

CASE 1: Processor P_j receives the pattern symbol P_{m-1} .

The packet with P_{m-1} does not include any attached messages. The received pattern symbol P_{m-1} is compared with T_j , and an attached message AM is created. If $P_{m-1} = T_j$, then AM is "POSSIBLE MATCHING (1)", otherwise it is "MISMATCH DETECTED (0)". Next, the symbol P_{m-1} is forwarded to P_{j-1} , if $j > 0$, while AM remains stored at P_j .

CASE 2: Processor P_j receives the symbol P_k . for $0 \leq k < m - 1$.

The packet carrying P_k . brings an attached message, say AM_{new} . There is also another attached message, say AM_{old} , residing at P_j . If AM_{new} is "POSSIBLE MATCHING", then T_j is compared with P_k and if $T_j = P_k$, then AM_{new} is changed to "MISMATCH DETECTED", otherwise it remains the same. Next, a packet storing P_k and AM_{old} is broadcast to P_{j-1} , provided that $j > 0$, while AM_{new} remains stored at P_j . It will be treated as AM_{old} after the symbol P_{k-1} has been received, for $k > 0$. If $k = 0$ then it is decided whether position j is the beginning of an occurrence of the pattern in the text or not. How this is done is described after the next lemma, from which the correctness of the algorithm follows.

Suppose a packet received by processor P_j brings a pattern symbol P_k and an attached message "POSSIBLE MATCHING". If $T_j = P_k$ then $T_{j+l} = P_{k+l}$ for any integer l such that both the inequalities $0 \leq j + l < n-1$ and $0 \leq k + l < m-1$ hold.

Proof: Induction on k . starting from $k = m - 1$ and then down to $k = 0$.

Now consider again the case when processor P_j (or S_j) receives the packet carrying the pattern symbol P_0 and an attached message AM . If both AM is "POSSIBLE MATCHING" and the equality $T_j = P_0$ holds, then, by lemma 1. the equalities $P_l = T_{j+l-1}$ hold for every integer l such that $0 \leq l \leq m - 1$ and $j + l - 1 \leq n - 1$. If this happens then it is said that pattern P is consistent with text T at position j . Clearly, a position j is the beginning of an occurrence of P in T if both P is consistent with T at j and $j + m - 1 \leq n - 1$. Hence, when the last packet with P_0 travels through the line, all the occurrences of the pattern P in the text T can be detected by verifying those two conditions. This completes the proof of the next result:

Algorithm APM finds all the occurrences of pattern P (or s) in text T on a array of $n = |T|$ processors in time $O(n)$.

Similarly APM find all the occurrences of pattern P in text T on array of m processors in time $O(n)$

Pattern Preprocessing

The string matching algorithm developed in this section consists of two stages: first is preprocessing of a. pattern, next is searching a text. In the preprocessing part the structure of a pattern is examined. The way the algorithm operates depends on

whether a pattern has a "short" or not. The short case is handled in a straightforward way; the long case is the key part of the algorithm.

Two Dimensional Pattern

In this section it is shown how a long period of a pattern can be found efficiently on a mesh, what is one of the main applications of the notion and properties of data partition. Let q be a data partition of a pattern P [5]. Visualize the word P as divided into k segments L_0, L_1, \dots, L_{k-1} of length q each, and then the segments put on top of each other, as in Figure 2.1

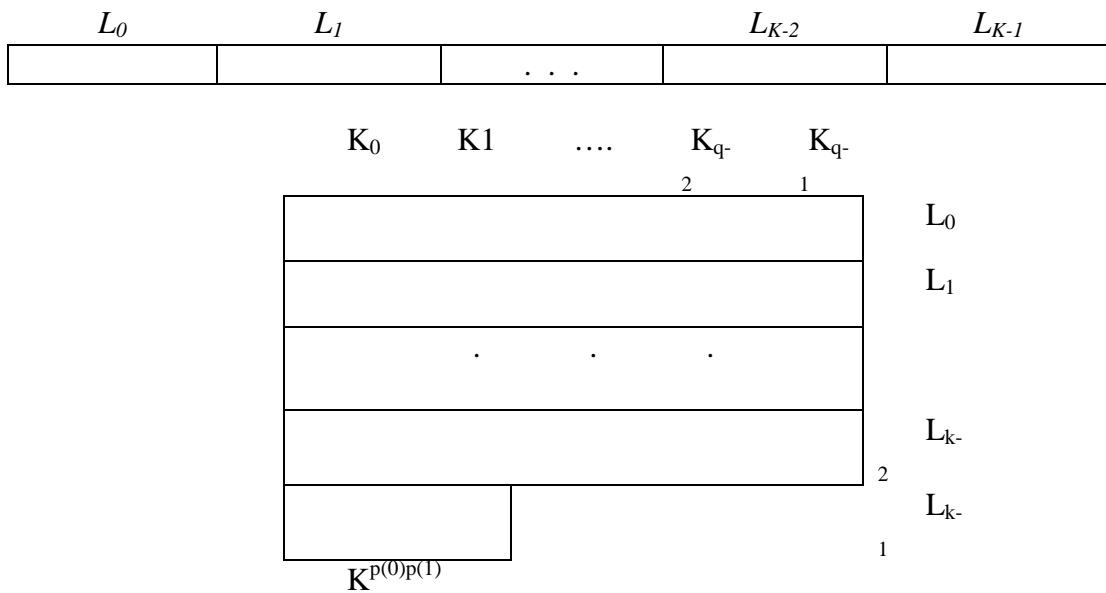


Figure 2.1: A data partition q of a pattern P determines a pattern array.

The last segment R_k may be of length smaller than q . This creates an array of size $q \times k$. it is called the pattern array. Notice that $k \leq \sqrt{n}$. The pattern array has columns $K_0 \dots K_{q-i}$, each of length k , as shown in Figure 2.1. Let p be the just found the period of P . In what follows witnesses for p will be needed, but only for positions of the form $i \cdot q$. They can be found during processing the pattern array to determine p .

Text Searching

The text-processing stage depends on the size of the pattern P . The pattern periodic case is solved analogously to known reduction of Breslauer and Galil [5]

Finding the Minimum Value for Pattern P Using M Processor

Let us consider input keys be k_1, k_2, \dots, k_m and then find the minimum of m key using M processors. Let T be the run time of this algorithm. We partition the input into \sqrt{m} part where each parts consists of \sqrt{m} key. Allocate each part to \sqrt{m} processor so that the minimum of each part can be computed in parallel

Algorithm Minimum(k_m)

```

{
    if (  $m=1$  ) return  $m$ 
    else
        divide  $m$  keys into  $\sqrt{m}$  part
         $k_1, k_2, \dots, k_m$ 
        similarly partition the processing so that  $p_i$  (  $1 \leq i \leq \sqrt{n}$  )
        let  $p_i$  find the minimum of  $k_i$  recursive
        if  $m_1, m_2, m_3, \dots, m_{\sqrt{m}}$  are the group minimum
        find the minimum
}

```

The maximum of m keys can be found in $O(\log \log m)$ time using n common CRCW PRAM Processors[15]

Decomposition of Pattern String

The string length distribute in a large range from a few bytes to hundreds in a parallel computing environment, which is infeasible to maintain in general for each string length based on the system resources availability[10,11]. We set a threshold t for any string longer than t that should be segmented into a set of substrings with length of t except the last sub string length is possible to shorter than the t . Two more bits are needed to store substring in memory. One bit is indicating for substring is partial match and another one is for entry valid. We also maintain two table such as Partial Match Table (PMT_{txt}) and Concatenate Verification Table (CVT)[20].

Pattern string size is large then divides into subparts like p_1, p_2, \dots, p_t and apply the following three operations such as AND, OR and NOT. The following algorithm shows for large string matching

A. Definition

A Subset pt_1 of a universe of discourse P is characterized by a membership

```

Function
     $f_p : a \rightarrow [0, 1]$ 
Algorithm Divide (P)
{
    If (P is small)
        Normal process
    Else
        Divide P into  $pt_1, pt_2, \dots, pt_l$ 
        Solve substrings in parallel
}

```

B. AND operation

The and operation is joining of two statements pt_1 and pt_2 is statement $pt_1 \text{ } pt_2$. the $pt_1 \text{ } pt_2$ truth value 1 wherever both pt_1 and pt_2 have the truth value 1, otherwise truth value 0.

$$pt_1 \text{ } pt_2 = 0 \quad \text{if } pt_i = 0$$

C. OR operation

The OR operation of two statements pt_1 and pt_2 is statement $pt_1 \vee pt_2$. the $pt_1 \vee pt_2$ truth value 0 when both pt_1 and pt_2 have the truth value 0, otherwise truth value 1.

$$pt_1 \vee pt_2 = 0 \quad \text{if } pt_i = 0$$

E. Algorithm for pattern matching

```

Algorithm(p)
{
  if a string is partial and the first substring register it in entry p
  else
  if a string is partial but not the first segment
  if the PMT entry p is valid
  Concatenate this string with content in entry p lookup the CVT
  if found in CVT
  if indicate this is the last block
  report match
  else
  replace the old entry in PMT with this string
  else
  drop it
  invalidate the entry p
  else
  drop it
}

```

F. Example

According to the general representation of string matching we consider the below example.

Text string: "RAMANA RAJA"

Pattern string: "RAJA"

Build a Boolean matrix by comparing text string with pattern string where '0' indicates unsuccessful match and '1' indicates successful match which is indicated in table 2. To get the string for the above example applies sparse technique from the obtained Boolean matrix. Mainly we are concentrating on two techniques such as decomposition of matrix and sparse [16,17].

Table 1: Sparse Table

0	1	2	3
2	5	1	5

For the above array apply sparse sort technique, and then find the least value. Now considering the least value's position in the array get the character position in the string. So here the least value is 1 by this we can get the character position "J" in the string.

Now assign a symbol to the obtained character position say ‘ β ’ and find the remaining character positions of the string by incrementing or decrementing the symbol based on string size. This process continues until the end of array. So in this example $\beta=2$ now apply $\beta-1$ and $\beta-2$ and $\beta+1$ we get the required string as “RAJA”. So the string matching is successful as shown in Table1.

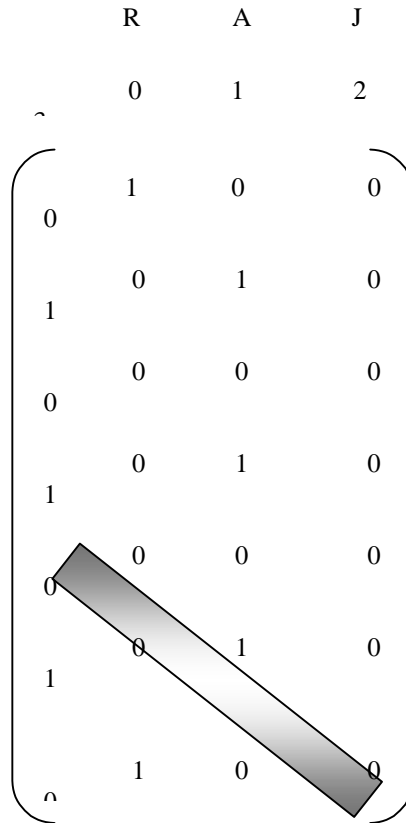


Figure 3.1: Identifying the string in the Boolean Matrix

For the above array apply parallel sort technique, and then find the least value. Now considering the least value’s position in the array get the character position in the string. So here the least value is 0. By this we can decide that string mismatch as shown in Table2.

Table 2: Sparse Table Indicating Unsuccessful Search

0	1	2	3
2	5	0	5

In the sparse table, smallest value is an upper bound (worst case) of approximate string matching and also we can identify the comparisons for string matching that is nothing but upper bound (pattern size * smallest value in sparse table) for preprocessed text.. The result as follows:

Let us assume the smallest value in sparse table K

The pattern string size m and text string size n

Now we can calculate the upper bound value (worst case) for finding of pattern string

Worst case = $K * m$ if $K \leq n/2$

All occurrences of a pattern in a text of length n can be found in time $O(\sqrt{K})$ on a $\sqrt{K} \times \sqrt{K}$ - mesh-connected computer.

Conclusions

The proposed strategy uses the knowledge from the given algorithm and mesh structure. We achieved a constant time (1) for mismatching. We obtain deterministic one and two dimensional arrays, which are probably the best in preprocessing and text, search. In this paper the main result is a $O(\sqrt{K})$ -time algorithm for $\sqrt{K} \times \sqrt{K}$ meshes with the help of mesh connected array of processors. The structure of an algorithm is based on the data partition technique. This is the first known optimal algorithm for pattern matching on meshes. We achieves a constant time $O(1)$ of any pattern string (as per the set S) for string mismatch and also can identify the upper bound($K \leq n/2$) for approximate string matching. Our future research is to do same problem on another model (such as omega, and hypercube), which contains minimum communication, and improve the performance.

Acknowledgements

We would like to thank our colleagues in the Department of Computer Science and Computer applications for their advice and helpful discussions. We thank Prof P.S.Raju, Director, and Dr.J.N.Murthy, Principal, GRIET for his moral support and providing Laboratory. We are grateful to the British Library, JNTUniversity Library for their services.

References

- [1] A.V. Aho, J.E. Hopcroft and J.D. Ullman, "The Design and Analysis of Computer Algorithms", Addison-Wesley Publishing Company. 1974.
- [2] T. Anderson, T. D.Culler, D.Patterson. "A case for NOW (Network of Workstations)", *IEEE Micro*, vol.15, 1995, pp.54–64.
- [3] M Allen,B.Wilkinson, "Parallel Programming: Techniques and Applications using Networked Workstations and Parallel Computers", *Prentice Hall*, 1999.
- [4] M.Crochemore W.Rytter, "Text Alogorithms", *Oxford University Press*, 1994.

- [5] E.Z.Galil and K.Park, "Truly alphabet-independent two-dimensional pattern matching", *Proc 33rd IEEE symp on Fund. of Computer science*, 1992, pp.247-256.
- [6] D.Gannon, W.Jalby, and K.Gallivan, "Strategies for cache and local memory management by global program transformation" *Journal on. Parallel and distributed computing*, vol.5,pp.587-616, 1998.
- [7] W.Gropp, E.Lusk, A.Skjellum. "Using MPI: Portable Parallel Programming with message Passing Interface", *The MIT Press, Cambridge, Massachusetts*, 1994.
- [8] Jin Hwan Park and K.M.George, "Parallel String Matching Algorithms based on dataflow", *Computer Science Department, Oklahoma State University, Stillwater,OK 74078, USA*.
- [9] F.T. Leighton, "Introduction to Parallel Algorithms and Architectures: Arrays, Trees, Hypercubes", 1992, Morgan Kaufmann, San Mateo, California.
- [10] P.Michailidis, K. Margaritis,"On-Line String Matching Algorithms: Survey and Experimental Results", *International journal of computer mathematics*, 2000.
- [11] P.Michailidis, K Margaritis, "String Matching Problem on a Cluster of Personal Computers: Experimental Results", *Proc. 15th International Conference Systems for Automation of Engineering and Research (SAER'2001)*, 2001.
- [12] W.Myers,W.Miller,"Approximate Matching of Regular Expressions", *Bulletin of mathematical biology*,Vol.51,No.1,pp 5-37,1989
- [13] P.Weiner,"Linear pattern matching algorithm", *Proc.14 IEEE Symposium on switching and automata*, pages 1-11,1973.
- [14] U.Vishkin,"Deterministic Sampling for fast pattern matching",1999.
- [15] S.Viswanadha Raju and A.Vinayababu "Performance in the design of Parallel Programming", *Proc ObComAPC-2004, Allied Publications*, pp.380 to 392.
- [16] S.Viswanadha Raju,A.Vinayababu and M.Mrudula , "Backend Engine for Parallel string Matching using Boolean Matrix" , *Proc PARELEC-2006,IEEE Computer Society*, pp 281-283.
- [17] S.Viswanadha Raju,S.R.Mantena,A.Vinayababu and GVSRaju, "Efficient Parallel String Matching Using Partition Method", *Proc PDCAT-2006,IEEE Computer Society-in press, TIAWAN*.
- [18] S.Viswanadha Raju, A.Vinayababu,S.P.Yanaiah and GVSRaju, "Parallel Approach for K String Matching", *Proc NCIMDiL-2006, Indian Institute Of Technology, Kharagpur, W.B. , pp: 5-10*.