

Systolic Algorithms For Tree Pattern Matching

Abdel Ejnoui and N. Ranganathan

Center for Microelectronics Research

Department of Computer Science and Engineering

University of South Florida, Tampa, Florida 33620

Abstract

The objective of tree matching is to find the set of nodes at which a pattern tree matches a subject tree. Several sequential and parallel algorithms have been proposed in the literature for this compute bound problem. Most of the parallel algorithms are based on the theoretical PRAM model of computation. In this paper, we propose two efficient parallel algorithms for tree pattern matching based on the linear systolic array model. The algorithms can be mapped onto any SIMD machine. The algorithms require $O(n+m)$ time to perform the matching using either n or m processors, where n is the size of the subject tree and m is the size of the pattern tree. The algorithms represent a significant improvement over the existing ones in view of implementation.

1. Introduction

Tree pattern matching is an important problem in computing used for 3-D object recognition in image understanding and vision systems. In 3-D object recognition, there are two major components: (i) the CAD-model which consists of 3-D volume representations of the boundaries of solid objects that are transformed into *model trees*, and (ii) the graylevel image which is preprocessed in order to separate overlapping objects in the image and the *object trees* for each object in the image has been extracted. The recognition procedure consists of two steps: (i) a hypothesis generation step in which the object tree of each object is matched to the model tree, (ii) a verification step in which each object is compared to a projection onto the plane of the model [2]. Tree pattern matching is also used in on-line interpreters and off-line applications such as code optimization in compilers, data-type specification and verification, and term rewriting systems.

Several sequential algorithms [4-7] and a few parallel algorithms [8-9] have been proposed in the literature for this computationally intensive problem. Most of the existing parallel algorithms are based on various PRAM models of computation. Smith and Lin have described a VLSI chip for tree matching in [10] which is the only hardware solution that could be found in the literature. Their solution suffers from some limitations which could be overcome in order to achieve a better solution. In this paper, we propose two efficient parallel algorithms based on a linear computational model for tree pattern matching. The algorithms are systolic in nature and can be mapped onto an SIMD computer or implemented as special purpose VLSI chips.

2. Problem Overview

The various definitions and the associated concepts are illustrated using the example in Figure 1. Tree pattern matching consists of finding all nodes in a subject tree at which a pattern tree matches. Let S be the subject tree and P be the pattern tree. The variables in P are represented by squares while the non variables are represented by circles. The variables appear as leaf nodes in the pattern tree. Our algorithms require the transformation of trees into strings as well as numbering the nodes of the trees in breadth-first manner.

Let Σ be an alphabet of function symbols including constants and V a set of variable symbols. We assume that $\Sigma \cup V = \emptyset$. Each function symbol f of Σ has a fixed unique arity $a(f)$ which corresponds the number of children of the node represented by f . A constant is a function symbol whose arity is 0 and corresponds to a leaf node.

Definition 1 A term is a symbol string defined recursively as: (i) Any constant or variable is a term, and (ii) If f belongs to Σ then $f(t_1, t_2, \dots, t_{a(f)})$ is a term given that each t_i is a term itself.

Definition 2 A term tree is an ordered labeled tree whose nodes are either symbols of Σ or V . A symbol of V will always appear as a leaf node in term trees.

Definition 3 The Euler Chain of a term tree, called EC , is a string whose elements are node numbers and is generated by visiting each node of the tree in a preorder depth-first sequence.

For example, in Figure 1, the Euler Chain of S (EC_S) is 12425859521363731 while the Euler Chain of P (EC_P) is 121343531.

Definition 4 The Euler String of a term tree, called ES , is a string whose elements are node labels and is generated by visiting each node of the tree in a preorder depth-first sequence.

In Figure 1, the Euler String corresponding to the tree represented by ES_S is ffbffafaffffbfaff while the Euler String corresponding to the tree represented by ES_P is fXffYfaff. Note that by substituting the node numbers in EC_S with node labels, we can derive the corresponding ES_S . Two term trees are equivalent if and only if their respective Euler Strings are identical. This is true since, if two Euler Strings are identical, their respective Euler Chains must also be identical. In Euler chains, each node can occur many times. Euler chains have the following properties with regard to tree structure: (i) a leaf node occurs only once in EC , (ii) the first and last occurrences of a node represents the root of a tree, and (iii) if a node has c children then this node occurs $c+1$ times in EC . In Figure 1, node 4 is a leaf and subsequently occurs only once in EC_S . The first and last occurrences of node 1 represent the root of S . Since node 1 has two children, it occurs three times in EC_S .

Let the number of nodes in the subject tree S be n and the number of nodes in the pattern tree P be m . It can be easily shown that the size of ES_S is $O(n)$ and the size of ES_P is $O(m)$. Let k be the number of occurrences of variables in ES_P and v_i the i th variable of ES_P . By dropping the variables from ES_P , a set of substrings is obtained which consists of non variable symbols.

Definition 5 σ_i is defined as a substring which consists of non variable symbols occurring between two consecutive variable symbols v_{i-1} and v_i .

In Figure 1, σ_1 , σ_2 and σ_3 represent the strings f, ff and faff respectively. Since there are k variables in ES_P , there must be $(k+1)$ such substrings in ES_P . Note that there are no variables to the left of σ_1 and to the right of σ_{k+1} in the example shown in Figure 1.

Definition 6 $EC_S(i)$ represents the node at the i th entry in EC_S from the left.

In Figure 1, $EC_S(8) = 9$ corresponds to node 9 which represents the symbol 'a'.

Definition 7 Occurrence Status (OS) is a 2-bit field, associated with each node, indicating first, last or other

occurrence of a tree node in EC_S . This field may have any one of the following four values:

Occurrence Status (OS)	Indication
0	unknown or middle occurrence
1	first occurrence
2	last occurrence
3	first and last occurrence

In Figure 1, node 1 occurs 3 times at the 1st, 11th and 17th entries from the left of EC_S . From the above definition, $OS(EC_S(1)) = 1$ and $OS(EC_S(17)) = 2$ and $OS(EC_S(11)) = 0$. On the other hand, node 4 occurs only once which means $OS(EC_S(3)) = 3$.

3. Proposed Approach

The proposed algorithms are based on a linear SIMD array architecture with bidirectional systolic data flow. Each processing element (PE) in the array requires simple hardware components such as an equality comparator, registers and control logic. Both algorithms consist of a simple preprocessing step followed by the pattern matching computation. The preprocessing step determines the first and last occurrences of each node in EC_S . In the matching step, the strategy is to find match positions for each σ_i within ES_S which will be combined to determine at different nodes the matches between the pattern and the subject. In this section, we describe the basic strategy used in the matching step of the proposed algorithm.

The matching of ES_P and ES_S is performed in the linear systolic array. We will first look at the matching of a single σ_i then extend it to the rest of the pattern. Let $\sigma_i = \sigma_{i,1} \sigma_{i,2} \dots \sigma_{i,l}$ be the substring of ES_P between the variables v_{i-1} and v_i , $s_j = s_{j,1} s_{j,2} \dots s_{j,l}$ be a substring of ES_S and $C_i = C_{i,1} C_{i,2} \dots C_{i,l}$ be the set of nodes in EC_S representing s_j , each having size l . Let $C_{i,0}$ and $C_{i,l+1}$ be the nodes of EC_S to the left and right of s_j respectively. With each node from $C_{i,0}$ through $C_{i,l+1}$ is associated a data structure as introduced in Definition 7. The fields of the data structure are (node x , position i , OS, last). Let $s_{i,l+1}$ and v_i be respectively the symbol of s_j and the variable both to the right of s_j in ES_S . The matching process consists of: (i) testing of partial matches, (ii) extraction of match parameters (MP), (iii) insertion of match parameters in the data flow and (iv) testing of legal substitutions and legal attachments. Each step is described below.

3.1 Testing of Partial Matches

Now, we state the conditions required to test the *partial matches* of each σ within ES_P which are for three cases:

- (i) σ_1 , (ii) σ_i where $2 \leq i \leq k$ and (iii) σ_{k+1} .

Case 1: σ_1 has a *partial match* at some position p of ES_S iff: (i) σ_1 matches ES_S for each symbol starting from p and (ii) $EC_S(p)$ and $EC_S(p+|\sigma_1|)$ are both first occurrences of nodes in EC_S .

Case 2 σ_i where $2 \leq i \leq k$ has a *partial match* at some position p of ES_S iff: (i) σ_i matches ES_S for each symbol starting from p , (ii) $EC_S(p-1)$ is the last occurrence of some node in EC_S and (iii) $EC_S(p+|\sigma_i|)$ is the first occurrence of some node in EC_S .

Case 3: σ_{k+1} has a *partial match* at some position p of ES_S iff: (i) σ_{k+1} matches ES_S for each symbol starting from p and (ii) $EC_S(p-1)$ and $EC_S(p+|\sigma_{k+1}|-1)$ are both last occurrences of nodes in EC_S . Let us assume the following configuration involving σ_j :

$$\begin{array}{ccccccc}
 C_{i,0} & C_{i,1} & C_{i,2} & \dots & C_{i,l} & & C_{i,l+1} \\
 & s_{i,1} & s_{i,2} & \dots & s_{i,l} & & s_{i,l+1} \\
 & \sigma_{i,1} & \sigma_{i,2} & \dots & \sigma_{i,l} & & v_i
 \end{array}$$

Now, each PE in the systolic array will hold one element of C_j , s_j and σ_j . In order to determine if each σ has a *partial match*, the following three conditions are tested within each PE:

- (1) if $s_{i,j} = C_{i,j}$ where $1 \leq j \leq l$ then condition 1 is true.
- (2) if ($i = 1$ and $OS(C_{i,1}) = 1$)
or ($i > 1$ and ($OS(C_{i,0}) = 2$ or $OS(C_{i,0}) = 3$))
then condition 2 is true.
- (3) if ($i = k+1$ and $OS(C_{i,l}) = 2$)
or ($i < k+1$ and ($OS(C_{i,l+1}) = 1$
or $OS(C_{i,l+1}) = 3$))
then condition 3 is true.

In the above, condition 1 tests for equality of the symbols which corresponds to situation (i) in Cases 1, 2 and 3. Next, in the case of σ_1 , the occurrence of the first node of EC_S is checked which corresponds to (ii) in Case 1. In the case of σ_{k+1} ($i = k+1$ in condition 3), the occurrence of the last symbol of σ_{k+1} is checked. This corresponds to situation (ii) in Case 3. In the case of σ_i where $2 \leq i \leq k$, the occurrence of the nodes to the left and to the right of σ is tested. This corresponds to (ii) and (iii) in Case 2.

3.2 Extraction of Match Parameters (MP)

For each substring σ_i when all the three conditions are met, certain information is stored in a data structure called *MP (Match Parameters)*. The data corresponding to σ_i is stored within the PE that corresponds to v_i . *MP* consists of five fields named as *MP1* through *MP5* which are defined as follows:

- MP1:* if $i < k+1$ then $MP1 = C_{i,l+1}$ else $MP1 = C_{i,1}$
MP2: if $i < k+1$ then $MP2 = i,l+1$ else $MP2 = i,1$
MP3: if $i < k+1$ then $MP3 = C_{i,l+1}.last$ else $MP3 = i,1$
MP4: $MP4 = |\sigma_i|$ or size of σ_i

MP5: $MP5 = i$ or index of σ_i

In the above, *MP1* contains the node to the right of σ_i , *MP2* contains the position within EC_S of that node's first occurrence (of its last occurrence in the case of σ_{k+1}), *MP3* contains the position of its last occurrence within EC_S , *MP4* contains the length of σ_i and *MP5* contains index of σ_i .

3.3 Insertion of Match Parameters

After an *MP* is detected in a PE, it must be inserted into a register and shifted to the next PE. During execution, *MPs* will move from one side to another across the array in a pipeline fashion. It is possible that during a cycle, a newly detected *MP* in a given PE may not be inserted into the pipe. The stage at this PE is already occupied by an *MP* which was detected several cycles earlier in a distant PE and has just been shifted from the immediate neighboring PE. We refer to this insertion conflict as a *collision*. Collisions will be resolved by using two pipes instead of a single one. An individual stage of a pipe consists of a shift register. Whenever there are no collisions, *MPs* are inserted in Pipe 1, otherwise they are inserted in pipe 2.

3.4 Detecting Legal Substitution & Legal Attachment

Let us assume that the match conditions were satisfied for two consecutive σ 's, say, σ_i and σ_{i+1} . It is important to note that *MP3* indicates the position of the last occurrence of the node to the right of σ_i which is equal to $(i+1,0)$. This also corresponds to the position of the node to the left of σ_{i+1} . Given this scenario, σ_i is matched with s_j symbol by symbol, v_i will be replaced by the substring of the subject from the $(i,l+1)$ th entry on the right of s_j to the $(i+1,0)$ th entry of the subject to the left of s_{i+1} , and σ_{i+1} will be matched with s_{i+1} symbol by symbol. If the matches are successful, the above results in what we describe as the *legal attachment* of substring $\sigma_i v_i$ to substring σ_{i+1} . The successful substitution of v_i by the subtree rooted at $C_{i,l+1}$ is referred to as a *legal substitution* as in [11]. Both are defined below.

Definition 10 A *legal substitution* of a variable v_i in ES_p is the replacement of v_i by a substring of ES_S where: (i) the leftmost and rightmost nodes of this substring are respectively the first and last occurrences of some node x in ES_S and (ii) σ_i has a *partial match* on the left of v_i which ends on the immediate predecessor node of the first occurrence of x .

Definition 11 A *legal attachment* of a substring $\sigma_i v_i$ onto σ_{i+1} is: (i) a *partial match* of σ_i ending at some position p in ES_S (ii) a *legal substitution* of v_i starting

from $p+1$ to some position q in ES_S and (iii) a *partial match* of σ_{i+1} starting at $q+1$ in ES_S .

In our configuration, the set of nodes in EC_S between the $(i,l+1)$ th entry and the $(i+1,0)$ th entry represents the Euler tour of the subtree of S rooted at the node corresponding at the $(i,l+1)$ th entry or at the $(i+1,0)$ th entry. These two entries represent the same node. During the algorithm execution, many matches for various σ s will occur. The purpose of tracking them is to test legal substitutions of each variable from v_j to v_k and legal attachment of each substring $\sigma_i v_i$ to the following $(\sigma_{i+1} v_{i+1})$ forming the overall match from $\sigma_1 v_1$ to $\sigma_{k+1} v_{k+1}$. If every test succeeds, then the pattern matches the subject at the node in which σ_1 starts matching ES_S . Now, the question arises as to how to detect legal substitutions and legal attachments.

Let us assume that $\sigma_i v_i \sigma_{i+1} v_{i+1}$ corresponds to a successful substitution and attachment in our configuration. As a result, the following statement must be true:

if $i < k$ then $(i+1,0) + |\sigma_{i+1}| + 1 = (i+1,l+1)$
 else $(i+1,0) + |\sigma_{i+1}| = (i+1,l)$.

Indeed since $(i+1,0)$ is the position in EC_S of the last node which replaced v_i , the term $(i+1,0) + |\sigma_{i+1}| + 1$ corresponds to the position of the first node to the right of σ_{i+1} within EC_S . This condition holds only if $i < k$. In the case of $i = k$, there is no variable immediately to the right of σ_{k+1} and consequently, $(i+1,0) + |\sigma_{i+1}|$ corresponds to the position within EC_S of the last node at which the pattern matches ES_S . This is the leftmost node at which σ_1 starts matching ES_S and represents the root of the subtree at which P matches S . By positioning ES_P at some node x in EC_S and by testing the above conditions for each $\sigma_i v_i$ from $i = 1$ to $i = k+1$, it becomes possible to determine if P matches S at node x . If this test is executed at each node of EC_S , we can find the set of all nodes where P matches S .

4. Systolic Algorithms

The two proposed systolic algorithms are described in this section. Both algorithms are based on the same computation within a PE. The difference is in the way the pattern and the subject strings flow within the array.

4.1 Systolic Algorithm 1

The systolic architecture for algorithm 1 is shown in Figure 2. The size of the linear array is $(m+2)$ where $m = |ES_P|$. The $(m+1)$ th PE serves as a dummy processor which contains the imaginary variable to the right of σ_{k+1} . This is to preserve the regularity of computation within all the PEs. The $(m+2)$ nd PE does not contain any pattern symbol, but includes a content addressable

memory (CAM) which is necessary for the matching computation. At the beginning of Algorithm 1, ES_P is loaded into the array from left to right and it remains stationary throughout the execution of the algorithm. ES_S and EC_S are loaded from the left side of the array with the rightmost node and symbol first, then shifted right to the next PE every clock cycle during execution.

The first $(m+1)$ PEs perform the same computation during each cycle. The CAM PE tests for legal attachment of each σ_i and detects matches in the tree. The MP s of each σ_i are right shifted each cycle through the array until they arrive at the CAM PE. Each entry in the CAM is an MP data structure and will be referred to as $CAM.MP$. A field such as $MP1$ within $CAM.MP$ will be referred to as $CAM.MP1$. The MP structures that arrive at the CAM PE through shifting will be referred to as MP and their corresponding fields such as $MP1$ will be referred to as $MP1$. The processing within the CAM PE at each cycle is shown in the form of pseudocode:

```
(C1) if  $MP5 = k+1$  then insert  $MP$  in CAM
(C2) else if  $MP5 = k$  then find  $CAM.MP$ 
      where  $MP3 + CAM.MP4 = CAM.MP3$ 
(C3) else find  $CAM.MP$ 
      where  $MP3 + CAM.MP4 + 1 = CAM.MP3$ 
(C4) endif
(C5) if  $CAM.MP$  is found then
(C6)  if  $MP5 = 1$  then declare match at  $CAM.MP1$ 
(C7)  else
(C8)     $MP1 = CAM.MP1$ 
(C9)    overwrite  $MP$  where  $CAM.MP$  is found
(C10) endif
(C11)endif
```

The first entry to be inserted into the CAM will be the MP corresponding to σ_{k+1} (C1). The statements (C2-C3) test if some given $\sigma_i v_i$ is legally attached to the next σ_{i+1} depending on the index of σ . When a *legal attachment* of two consecutive σ s is detected (C5) and the index field $MP5 = 1$ then, the pattern matches the subject at the node whose MP is stored at $CAM.MP1$ (C6). Else, the newly arrived MP overwrites the found CAM entry after the node value contained in $CAM.MP1$ is transferred to $MP1$ (C8-C9). The reason behind this transfer is to preserve the node stored in the MP of σ_{k+1} . This node is actually the root node of the subtree at which the pattern tree matches the subject tree.

It is important to note that whenever σ_{k+1} has a partial match at a new position p of ES_S , the resulting MP is inserted in a new entry of the CAM. The node in this MP represents the root of the subtree at which the pattern starts matching the subject. When σ_k legally attaches onto σ_{k+1} , its corresponding MP is inserted in the same

CAM entry. The process is repeated for each subsequent σ that legally attaches onto its right neighbor. If one σ fails legally to attach at some point during execution, then the CAM entry used for insertion becomes obsolete for the match at position p . This means that the pattern will not match the subject between the node at position p of ES_S (which is a last occurrence) and its first occurrence in ES_S . This is why we insert the MP of every partial match of σ_{k+1} in a new CAM entry. It should be noted that the size of the CAM is at most $O(\log n)$ where n is the size of the subject tree.

Let $ES_S = n$ and $ES_P = m$, then the time required to execute Algorithm 1 is: $T = 2n + m + 2$. It will take n cycles to load the subject and the pattern concurrently and $n + m$ to match them.

4.2 Systolic Algorithm 2

The systolic array for algorithm 2 consists of n PEs where $|ES_S| = |EC_S| = n$. All PEs are similar in structure. Each PE will contain a node of EC_S with its associated data structure (node x , position i , OS, last), a symbol of ES_S , and a symbol of ES_P with its associated data structure (symbol, position, vflag, size, index). At the beginning of Algorithm 2, EC_S and ES_S are loaded into the systolic array from left to right and remain stationary throughout the execution of the algorithm. ES_P is loaded into the array from right to left with the leftmost symbol first, then shifted left to the next PE every clock cycle.

All PEs execute the same computation at each clock cycle. Whenever there is a legal substitution by a variable, its match parameters are written into the MP data structures and shifted left to the next PE. The legal substitution of v_j is treated differently. If v_j successfully replaced a subtree in the subject, its match parameters are written into its MP data structure but will not be shifted to the left. The corresponding MP will remain in the same PE where it was recorded. As the MP corresponding to the match of σ_2v_2 arrive from the right, it will be used to test if there is a legal attachment of σ_1v_1 and σ_2v_2 . If it was successful, the match parameters of σ_2v_2 replace those of σ_1v_1 . As the MP corresponding to other σ_1v_i substrings arrive from the right they are tested for legal attachment. The above process is repeated for each σ_1v_i . If there is a legal attachment for every σ , then there is a match at the node contained within the match parameters of σ_{k+1} .

Let $ES_S = n$ and $ES_P = m$, then the time required to execute Algorithm 2 is: $T = 2n + m$. It will take n cycles to load the subject and the pattern concurrently and $n + m$ to match them. The details of both systolic algorithms and their analysis can be found in [3].

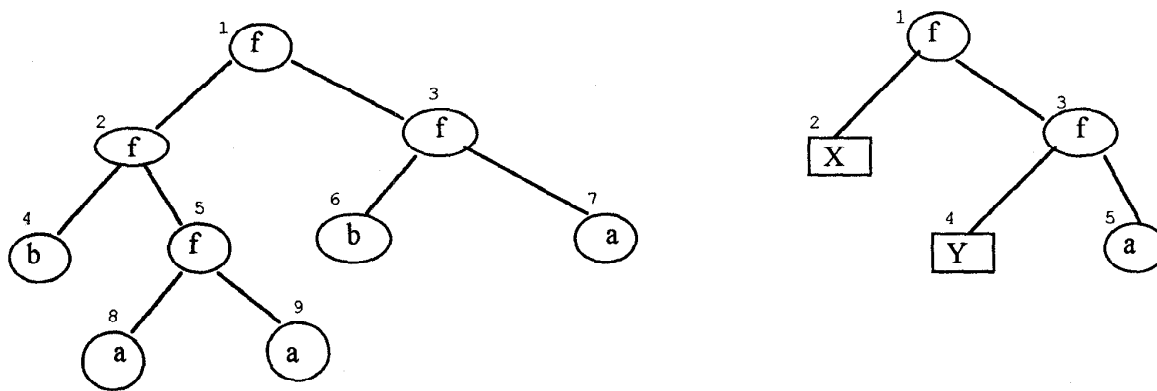
5. Performance Comparison and Conclusions

Table 1 provides a general comparison of the different algorithms in terms of processing time, number of processors and computational model. The main advantage of our algorithms is that they can be efficiently implemented in any SIMD parallel machine or can be realized as special purpose VLSI chips. The computations as well as the logic within each PE is very simple requiring only a comparator, some registers and associated control logic. In order to maintain a small clock cycle for the chip, the CAM may be partitioned into smaller sizes if required.

Acknowledgement: This work is supported in part by a National Science Foundation Grant No. MIPS-9407034 and also in part, by the ARPA, under NASA cooperative agreement #NCC8-31 (federal support provided by NASA Marshall Space Flight Center Technology Reinvestment Project and matching funds provided by Enterprise Florida Inc. and Dynacs Engineering Company).

References

- [1] K. Tarora, T. Hirata, Y. Inagaki, *A Parallel Algorithm for Tree Pattern Matching*, Systems and Computers in Japan, 24, 5, 1993.
- [2] E. Gmur, H. Bunke, *3-D Object Recognition Based on Subgraph Matching in Polynomial Time*, Structural Pattern Analysis, World Scientific, 1990.
- [3] A. Ejnoui, *Parallel Algorithms for Tree Pattern Matching*, M.S. Thesis, U. of S. Florida, Mar 1995.
- [4] C. M. Hoffman, M. J. O'Donnell, *Pattern Matching in Trees*, JACM 29, 1, Jan. 1982.
- [5] M. Dubiner, Z. Galil, E. Magen, *Faster Tree Pattern Matching*, Proc. 31st IEEE Symp. on FOCS, 1990.
- [6] K. Kojima, *A Pattern Matching Algorithm in Binary Trees*, Lecture Notes in Computer Sci., 147, 1983.
- [7] S. R. Kosaraju, *Efficient Tree Pattern Matching*, Proc. 30th IEEE Symp. on FOCS, 1989.
- [8] R. Ramesh, R. M. Verma, T. Krishnapasad, I. V. Ramakrishnan, *Term Matching on Parallel Computers*, J. of Logic Programming, 6, 3, 1989.
- [9] R. Ramesh, I. V. Ramakrishnan, *Parallel Tree Pattern Matching*, J. of Symbol. Comput., 9, 4, 1990.
- [10] Smith, J. Lin, *Tree Match Chip*, IEEE Transactions on Computers, 40, 5, 1991.
- [11] R. Ramesh, I. V. Ramakrishnan, *Nonlinear Pattern Matching in Trees*, JACM 39, 2, 1990.
- [12] R. Ramesh, I. V. Ramakrishnan, *Optimal Speedups for Parallel Pattern Matching in Trees*, Proc. Rewriting Techniques and Applications, Bordeaux, France, 1987.



(a) Subject Tree S (b) Pattern Tree P
 Figure 1. An example of subject and pattern trees.

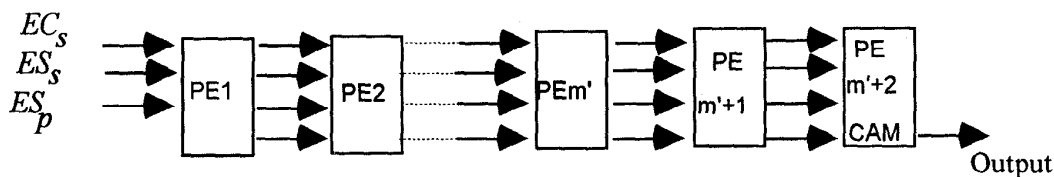


Figure 2. Systolic array for algorithm 1

n = size of subject tree; m = size of pattern tree; k = number of variables in the pattern tree

Algorithms	Time Complexity	Number of Processors	Computational Model
Bottom-up [4]	$O(2^{m+n})$	1	Sequential
Top-down [4]	$O(mn)$	1	Sequential
Kosaraju [7]	$O(nm^{0.75} \text{polylog}(m))$	1	Sequential
Dubiner et al. [5]	$O(nm^{0.5} \text{polylog}(m))$	1	Sequential
Ramesh et. al [11]	$O(m + nk)$	1	Sequential
Ramesh et. al : alg. 1 [9,12]	$O(n \log n)$	$O(n^{2-\epsilon})$	EREW PRAM
Ramesh et. al : alg. 2 [9,12]	$O(\log^2 n)$	$O(nk/\log^2 n)$	CREW PRAM
Tarora et al [13]	$O(\log n)$	$nm/\log n$	CREW PRAM
Smith & Lin [10]	$O(m+n)$		VLSI
This work:			
Proposed Algorithm 1	$O(m+n)$	m	Systolic
Proposed Algorithm 2	$O(m+n)$	n	Systolic

Table 1. Summary of tree pattern matching algorithms