A Parallel Algorithm and Architecture for Object Recognition in Images

K. Sitaraman, A. Ejnioui and N. Ranganathan

Dept of Computer Science and Engineering
University of South Florida
4202 E Fowler Avenue, ENB 118
Tampa, FL 33620
Email: ranganat@csee.usf.edu
Phone: 813 974 4760
Fax 813 974 3610

0-7803-7970-5/03/$20.00 ©2003 IEEE

Authorized licensed use limited to: New Jersey Institute of Technology. Downloaded on March 24,2010 at 12:27:47 EDT from IEEE Xplore. Restrictions apply.

A Parallel Algorithm and Architecture for Object Recognition in Images

Abstract

The problem of tree pattern matching for object recognition in images is computationally intensive in nature. In two-dimensional images, the objects can be represented through multiscale decomposition as tree structures. The pattern tree representing an object can be matched with a subject tree representing an image in order to detect the objects within the image. Several sequential, parallel and hardware algorithms exist in the literature for tree pattern matching. In this paper, we describe a new parallel algorithm and its realization as a VLSI chip for tree pattern matching. The hardware algorithm is based on a linear array of processing elements (PEs) where the pattern matching is done in a pipelined fashion relying on nearest-neighbor communication between the PE's and the subject and pattern trees of arbitrary length can be processed using a fixed size PE array. The algorithm has an improved execution time of $O(\lceil m/a \rceil n)$ required to perform the matching where m, a and n are the sizes of the pattern tree, processor array, subject tree respectively. A prototype CMOS VLSI chip implementing the proposed algorithm has been designed and verified. It is shown that the hardware algorithm proposed in this work represent a significant improvement in terms of computational complexity, data flow, and architecture over the ones previously proposed for this problem.

1. Introduction

A tree structure can be used to describe the evolution of the contour of an object at increasing levels of detail, from the root down to the leaves. The contour shapes are represented at all scales by sequences of alternating concave/convex segments identified by inflection points. Each tree node represents a contour segment of the object by a set of attributes. As an example, Figures 1 through 5 show how the image of an airplane is represented with a tree. Figures 1 - 4 show how the boundary of an airplane is successively divided into subregions based on the inflexion points, and how image resolution increases from one image to the other. The nodes of the tree correspond to the boundary segment and arcs connect nodes at successive levels of resolution. The children of a given node describe the structural change affecting a given segment between consecutive resolution levels. The children nodes divide the segment represented by the parent node and provide better detail about the sub-segment.

An image database can be constructed by combining trees of various objects under a single root node. A subject tree can represent an image from the database while a pattern tree can represent an unknown image, which needs to be compared to the images stored in the database. By matching both trees, the nodes at which the pattern tree matches the subject tree are identified. The matching nodes correspond to the database images that match the unknown image. The pattern tree can be tailored in order to obtain a complete match, a coarse match, to occlude a detail of the image, or to match only a particular feature of the image. This is done with the help of a special type of node, called *variable node*, in the pattern tree, which is capable of substituting any subtree. In the context of normalization, tree pattern matching appears in many applications such as the evaluation of input expressions in functional languages [2, 3], online interpreters [4], compiler code optimization [5], theorem proving of word problems [6], data-type specification and verification in software validation [7], and computer algebra [8] although our motivation arises from the problem of object recognition from images.

In this paper, we propose a new algorithm, and its realization as a systolic algorithm and VLSI chip for tree pattern matching. We discuss the related wor he next section and then present the details of the proposed algorithm, architecture, and the chip implementation in the following sections.

0-7803-7970-5/03/$20.00 ©2003 IEEE

Authorized licensed use limited to: New Jersey Institute of Technology. Downloaded on March 24,2010 at 12:27:47 EDT from IEEE Xplore. Restrictions apply.

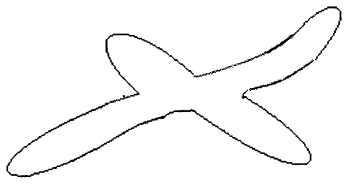


Figure 1. Coarse Image, Level 0.

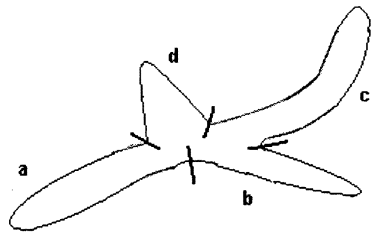
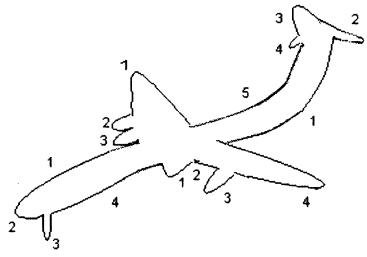


Figure2. Division of Contour, Level 1.



Figures 3. Further Division, Level 2.

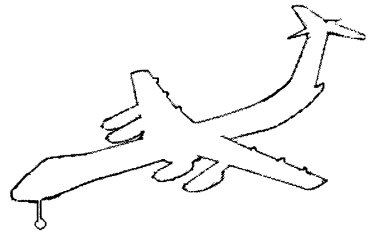


Figure 4. Final Image, Leaf level.

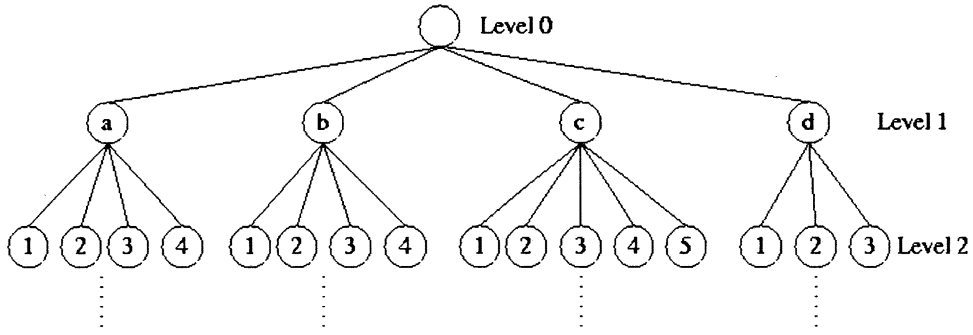


Figure 5. Tree representation of an airplane.

2. Related Work

Several sequential algorithm [12] [13] [14] [15] and parallel algorithms [16] [17] [18] have been proposed for this computationally intensive problem. The majority of the parallel algorithms are based on the PRAM model of computation. An earlier hardware solution to this problem has been proposed as a chip implementation rich in content-addressable memories, shift registers and bitstacks where a bitstack is used to contain a subtree that can be replaced by a variable [19]. Since each of these bitstacks have a limited size, it may not be able to contain an entire large subtree candidate for replacement and as a result one or more partial matches among all possible partial matches may be missed. Another hardware solution has been proposed in which the PE array consists of one variable processing PE (VPE) and several symbol processing PEs (NPE) [20]. This architecture imposes two limitations by which (i) it can handle only pattern substrings that fit into the PE array and (ii) it can handle only one pattern substring at a time. This architecture can be improved by removing both limitations. However, this may lead to a situation where match parameters generated upstream in the array start to shift from left to right across the array, then later may collide with other match parameters generated downstream in the array. In this paper, we propose a new hardware algorithm that removes the two limitations mentioned above and

Authorized licensed use limited to: New Jersey Institute of Technology. Downloaded on March 24,2010 at 12:27:47 EDT from IEEE Xplore. Restrictions apply.

eliminates the collision problem associated with traveling match parameters. In addition, this algorithm requires significantly thinner architecture and data flow.

3. Preliminaries

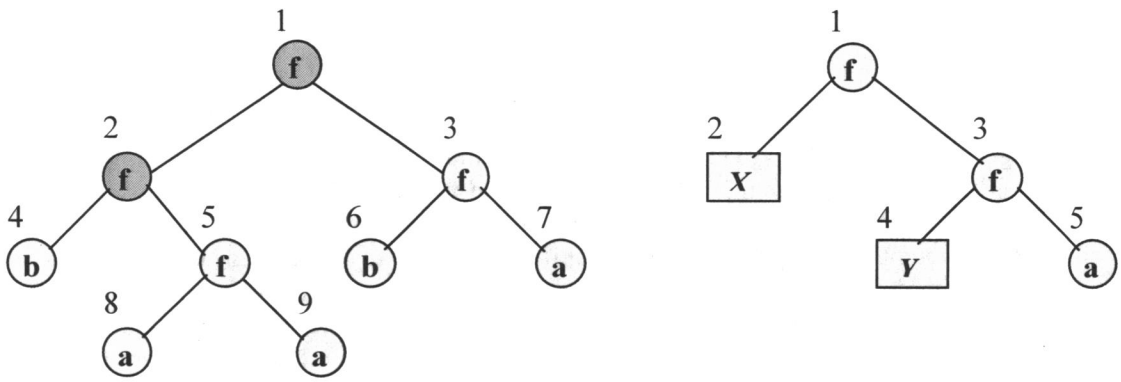
We present in this section definitions and terminology associated with the tree pattern matching problem using the trees shown in Figure 6. Let S and P be the subject and pattern trees respectively. The variables in P are enclosed in squares while the symbols are enclosed in circles. The variables can only be leaves in P. Processing these trees requires their transformation into strings. Let Σ be an alphabet of function symbols including constants and V be a set of variable symbols where $\Sigma \cap V = \varnothing$. Each function symbol f of Σ has a fixed arity $a(f)$ that represents 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.

Term: A term is a symbol string defined recursively as: (i) any constant or variable is a term, and (ii) if $f \in \Sigma$, then $f(t_1, t_2, \ldots, t_{a(f)})$ is a term given that each t_i is a term itself.

Tree: A term tree is a labeled directed tree $T(r, N)$ such that: (i) N is a finite set of nodes, (ii) r is the root which has indegree 0 and every other node has indegree 1, (iii) if a node is a variable, it has outdegree 0, (iv) if a node is a symbol, it has outdegree 0 or more.

Euler Chain: The Euler Chain (C) of a term tree $T(r, N)$ consisting of the subtrees $T_1(r_1, N_1)$, $T_2(r_2, N_2)$, …, $T_q(r_q, N_q)$, is the string $re_1re_2r\ldots re_q$ where e_i is the Euler Chain of the subtree $T_i(r_i, N_i)$ rooted at r_i. In Figure 6, the Euler Chain of S (C_S) is 12425859521363731. For a given term tree, its Euler Chain C has the following properties: (i) a leaf node occurs only once in C, (ii) the first and last occurrences of a node represent the subtree rooted at that node, (iii) if a node has a children, then this node occurs $a + 1$ times in C. In Figure 6, node 4 in S is a leaf and subsequently occurs only once in C_S while node 1 is a root with two children and subsequently occurs three times in C_S.

Euler String: The Euler String (E) of a term tree is obtained by replacing all the nodes in the Euler Chain by their labels. In Figure 6, the Euler String of S (E_S) is *ffbffafafffffbfaff*. Let $C(i)$ and $E(i)$ be the number and the label of the ith node from the left of C and E respectively. For example, $C_S(6) = 8$ and $E_S(6) = a$. Our algorithm verifies whether the Euler Strings of the subject and pattern trees are identical after replacing the variables in the pattern by Euler Strings of appropriate terms. Two term trees are equivalent iff their Euler Strings are identical. Let k be the number of variables in P and v_i the ith variable in P.



Subject Tree S Pattern Tree P

C_S = 12425859521363731
E_S = *ffbffafafffffbfaff*

C_P = 121343531
E_P = *fXffYfaff*

Figure 6. An example of a subject and pattern tree.

Authorized licensed use limited to: New Jersey Institute of Technology. Downloaded on March 24,2010 at 12:27:47 EDT from IEEE Xplore. Restrictions apply.

Pattern Substring: σ_i is the substring which consists of non variable symbols occurring between two variable symbols v_{i-1} and v_i in E_P. In Figure 6, σ_1, σ_2, and σ_3 represent the substrings *f*, *ff*, and *faff* respectively. Let $\sigma_i(j)$ be the *j*th symbol from the left of σ_i. For example, $\sigma_3(2) = E_P(7) = a$ whose $C_P(7) = 5$. Since there are k variables in E_P, there must be $k + 1$ such substrings in E_P with no variable to the left of σ_1 and to the right of σ_{k+1} respectively.

Occurrence Status: The occurrence status of a node (*OS*) is a label associated with each node indicating the first, last or otherwise occurrences of that node in C and can have the following values: (i) *FIRST* if this is its first occurrence, (ii) *LAST* if this is its last occurrence, (iii) *FIRSTLAST* if this is simultaneously its first and last occurrence, (iv) *NONE* if its occurrence is otherwise. For example, node 1 occurs three times at the 1st, 11th, and 17th positions from the left of C_S. As a result, $OS(C_S(1)) = OS(E_S(1))$ = FIRST, $OS(C_S(17)) = OS(E_S(17))$ = LAST and $OS(C_S(11)) = OS(E_S(11))$ = NONE. On the other hand, node 4 occurs only in the 3rd position from the left of C_S and as a result $OS(C_S(3)) = OS(E_S(3))$ = FIRSTLAST.

4. Basic Approach

In this section, we explain the theoretical approach behind our algorithm. Let $\sigma_i = \sigma_i(1)\sigma_i(2)\ldots\sigma_i(j)$ be a substring of symbols between v_{i-1} and v_i in E_P, $E' = E(p)E(p+1)\ldots E(p+j-1)$ be a substring of E_S and $C' = C(p)C(p+1)\ldots C(p+j-1)$ be its corresponding Euler subchain each having size j. We can assume that $C(p-1)$ and $C(p+j)$ are the nodes to the left of $C(p)$ and to the right of $C(p+j-1)$ respectively. Similar assumption can be applied to $E(p)$ and $E(p+j-1)$. The matching process executed by the algorithm consists of: (i) testing of *partial matches*, (ii) identification of *legal substitutions*, (iii) identification of *legal attachments*, and (iv) identification of *matches*.

4.1. Testing of Partial Matches

We state the conditions required to test the *partial matches* for each σ_i within E_S depending on σ_i's location in E_P as follows:

- σ_1 (leftmost σ in E_P): σ_1 has a *partial match* at some position p of E_S iff: (i) $\sigma_1(l) = E_S(q)$ for (l, q) where $1 \le l \le j$ and $p \le q \le p+l-1$, (ii) $OS(C(p)) = OS(C(p+j))$ = FIRST or FIRSTLAST. This means that σ_1 must match E_S for each symbol starting from position p and the occurrences of the first node in C' and the node to the right of the last node in C' are both first occurrences in C_P. In Figure 7, σ_1 has a partial match on positions 1, 2, and 5 in E_S.
- σ_i for $2 \le i \le k-1$ (anywhere between leftmost and rightmost σs of E_P): σ_i has a *partial match* at some position p of E_S iff: (i) $\sigma_i(l) = E_S(q)$ for (l, q) where $1 \le l \le j$ and $p \le q \le p+l-1$, (ii) $OS(C(p-1)) = OS(C(p+j))$ = FIRST or FIRSTLAST. This means that σ_i must match E_S for each symbol starting from position p, the occurrence of the node to the left of the first node in C' is its last occurrence in C_P, and the occurrence of the node to the right of the last node in C' is its first occurrence in C_P. In Figure 7, σ_2 has a partial match on positions 4, 10, and 11 in E_S.
- σ_{k+1} (rightmost σ in E_P): σ_{k+1} has a *partial match* at some position p of E_S iff: (i) $\sigma_{k+1}(l) = E_S(q)$ for (l, q) where $1 \le l \le j$ and $p \le q \le p+l-1$, (ii) $OS(C(p-1)) = OS(C(p+j-1))$ = LAST or FIRSTLAST. This means that σ_{k+1} must match E_S for each symbol starting from position p and the occurrences of the first node to the left node in C' and the last node in C' are both last occurrences in C_P. In Figure 7, σ_3 has a partial match on positions 7 and 14 in E_S.

$OS(C_S)$ = F (FIRST) or L (LAST) or FL (FIRSTLAST) or N (NONE)
PM = Partial Match
$E_P = f\underline{X}ff\underline{Y}faff$

Authorized licensed use limited to: New Jersey Institute of Technology. Downloaded on March 24,2010 at 12:27:47 EDT from IEEE Xplore. Restrictions apply.

Position in E_S	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
C_S	1	2	4	2	5	8	5	9	5	2	1	3	6	3	7	3	1
$OS(C_S)$	F	F	FL	N	F	FL	N	FL	L	L	N	F	FL	N	FL	L	L
E_S	*f*	*f*	*b*	*f*	*f*	*a*	*f*	*a*	*f*	*f*	*f*	*f*	*b*	*f*	*a*	*f*	*f*
σ_1 (*f*)	P M	P M			P M												
σ_2 (*ff*)				P M						P M	P M						
σ_3 (*faff*)							P M							P M			
Match 1	*f*	$\underline{X}$	←	—	—	—	—	—	→	$\underline{X}$	*f*	*f*	*f*	$\underline{Y}$	*f*	*a*	*f*	*f*
Match 2		*f*	$\underline{X}$	*f*	*f*	$\underline{Y}$	*f*	*a*	*f*	*f*							
Failed Match					*f*	$\underline{X}$	∇										

Figure 7. Partial matches, legal substitutions, attachments of subject and pattern trees shown in Figure 6.

4.2. Identification of Legal Substitutions

If two consecutive pattern substrings have partial matches, then the variable between them may possibly be replaced by a subtree, which can be represented by a string. Let $E' = E(x)E(x+1)\ldots E(x+j-1)$ be a substring of E_S and $C' = C(x)C(x+1)\ldots C(x+j-1)$ be its corresponding Euler subchain each having size j. A *legal substitution* is the replacement of a variable v_i in E_P by E' iff: (i) σ_i has a partial match at some position p of E_S and ends at $E(x-1)$ where $p < x$, (ii) $E(x) = E(x+j-1) = y$ where y is some node in S, (iii) $OS(C(x)) =$ FIRST and (iv) $OS(C(x+j-1)) =$ LAST. This means that v_i can replace the substring E' only if σ_i, located to its left, has a partial match that ends on the immediate predecessor node ($E(x-1)$) of some node y in E_S, and the leftmost and rightmost nodes of this substring E' are respectively the first and last occurrences of the same node y in E_S. E' represents a subtree rooted at node y. Note that it is possible to have a legal substitution involving a one-node subtree and the four conditions stated above will still hold. In Figure 7, variable X is involved in a legal substitution where it can be replaced by the substring starting at position 2 and ending at position 10 in E_S. This substring is the subtree rooted at node 2 in the subject tree S shown in Figure 6. On the other hand, variable Y is involved in another legal substitution where it can be replaced by the substring consisting of only one symbol in position 13 in E_S. This substring represents the one-node subtree rooted at node 6 in the subject tree S shown in Figure 6.

4.3. Identification of Legal Attachments

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 E_S, (ii) a legal substitution of v_i starting from $(p + 1)$ to some position q in E_S, (iii) a partial match of σ_{i+1} starting at position $(q + 1)$ in E_S. In Figure 7, σ_1 has a partial match at position 1 in E_S, variable X has a legal substitution with the substring starting from position 2 to 10 in E_S, and σ_2 has a partial match starting from position 11 in E_S. The attachment of $\sigma_1 X$ onto σ_2 is a legal attachment.

4.4. Identification of Matches

When each substring $\sigma_i v_i$ legally attaches to σ_{i+1} for all $1 \le i \le k$ starting from some position p in E_S, a

Authorized licensed use limited to: New Jersey Institute of Technology. Downloaded on March 24,2010 at 12:27:47 EDT from IEEE Xplore. Restrictions apply.

match occurs where E_P matches E_S at that position p. This is equivalent to the pattern tree P matching the subject tree S at the node in position p in E_S. In Figure 7, $\sigma_1 X$ legally attaches to σ_2 starting from position 1 in E_S, and $\sigma_2 Y$ legally attaches to σ_3. As a result, E_P matches E_S at position 1, which is equivalent to P matching S at the shaded node 1 of S shown in Figure 6.

5. Hardware Algorithm

The proposed algorithm is based on a linear SIMD array architecture, with bi-directional data flow, and a First In First Out (FIFO) buffer shown in Figure 8. The array is first loaded from the left with E_P. Then both E_S and E_P are fed concurrently at every other clock cycle from the left. E_P remains stationary in the array. As a result, a PE may contain either a symbol or a variable from E_P. As the subject is pushed across the array, each PE executes a specific computation whose result is represented by a single bit called flag bit. This flag bit is shifted at each clock cycle from the rightmost PE leftward. If at some point, it reaches the leftmost PE, a match is declared between the subject and the pattern. The FIFO buffer is used when the pattern does not fit completely into the PE array and partitioning is required.

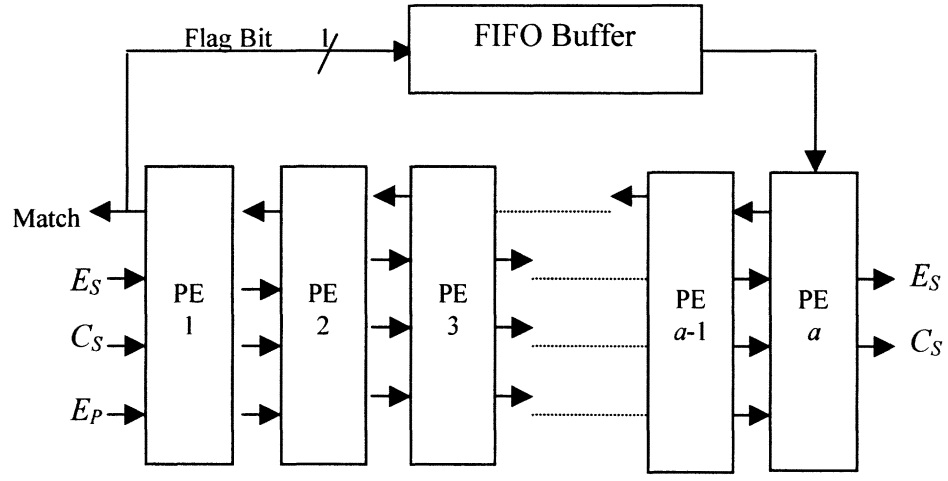


Figure 8. Linear PE array for the tree pattern matching problem.

5.1. Computation in Symbol-Containing PEs

Let a PE contain a symbol in position i of E_P. Let this symbol be $E_P(i)$. This symbol must belong to some pattern substring σ in some position q. At the current clock cycle, this PE also contains a symbol in position j of E_S. Let this symbol be $E_S(j)$ and its corresponding $C_S(j)$. Let b be the incoming flag bit from the right neighboring PE. This PE will execute the following computation:

```
if b = 0          // test the incoming bit
        shift 0 to the left PE
        stop
endif
if E_P(i) != E_S(j) // test if the pattern and subject symbols are identical
        shift 0 to the left PE
        stop
endif
if i = 1 then     // test if the pattern symbol is first in the pattern string
        if OS(C_S(j)) != FIRST then     //     f this is the first occurrence of the subject symbol
                shift 0 to the left PE
                stop
```


Authorized licensed use limited to: New Jersey Institute of Technology. Downloaded on March 24,2010 at 12:27:47 EDT from IEEE Xplore. Restrictions apply.

```
            endif
        endif
        if i = m then     // test if the pattern symbol is last in the pattern string
            if OS(C_S(j)) != LAST then        // test if this is the last occurrence of subject symbol
                shift 0 to the left PE
                stop
            endif
        endif
        if q = 1 then     // test if this the first symbol in the pattern substring
            // test if this is the last occurrence of the subject symbol to the left of this subject symbol
            if OS(C_S(j-1)) != LAST and OS(C_S(j-1)) != FIRSTLAST then
                shift 0 to the left PE
                stop
            endif
        endif
        if this PE is the leftmost PE in the array
            declare a match at position j in E_S
        endif
        shift 1 to the left PE
```

5.2. Computation in Variable-Containing PEs

Let a PE contain a variable v_i in E_P. At the current clock cycle, this PE also contains a symbol in position q of E_S. Let this symbol be $E_S(q)$ and its corresponding $C_S(q)$. Assume that $E_S(j) = E_S(q)$ where $j \leq q$ and $OS(E_S(j)) = OS(C_S(j))$ = FIRST. j is the position of the first occurrence of the same node in positions j and q in E_S. Let b be the incoming flag bit from the right neighboring PE. This PE will execute the following computation:

```
        if OS(C_S(q)) = NONE then                // test if the occurrence of subject symbol is none
            shift 0 to the left PE
        else if OS(C_S(q)) = FIRST then  // test if the occurrence of subject symbol is first
            if last stored symbol = C_S(q)    // the variable has been successfully replaced by a
subtree
                shift 1 to the left PE
                remove last stored symbol
            else
                shift 0 to the left PE
            endif
        else if OS(C_S(q)) = LAST then   // test if the occurrence of the subject symbol is last
            shift 0 to the left PE
            if b = 1 then
                store symbol     // last symbol in the subtree euler string has to be stored
            endif
        else if OS(C_S(q)) = FIRSTLAST then    // test if the occurrence of the subject symbol is firstlast
            if b = 1 then
                shift 1 to the left PE     // the variable has been replaced by a single node
subtree
            else
                shift 0 to the left PE
            endif
        endif
```


Authorized licensed use limited to: New Jersey Institute of Technology. Downloaded on March 24,2010 at 12:27:47 EDT from IEEE Xplore. Restrictions apply.

This PE saves the position in E_S of the first occurrence of its current subject symbol. Then it compares this position with that of every incoming subject symbol until the two positions are equal, in which case it shifts the 1-bit to the left. The maximum number of symbols to be stored will be equal to the height of the subject tree.

5.3. Partitioned version of the Algorithm

Since every PE array has a fixed size, a partitioning scheme is devised to handle the case where the pattern cannot fit in the array. E_P is partitioned into substrings large enough to fill the entire array. We start processing the rightmost substring of E_P, then process the ones to its left and so on. A FIFO buffer, located at one end of the PE array, is needed to store temporarily the flag bit that travels across the array from right to left. The partitioning scheme is as follows:

```
for each substring of E_P starting from the rightmost one
        shift the match bit from the FIFO buffer to the rightmost PE
        load the substring in the PE array
        start pushing E_S and C_S left to right every other clock cycle
        execute the hardware algorithm for the entire E_S and C_S
        shift the match bit from the leftmost PE into the FIFO buffer
endfor
```

A trace of the algorithm is shown below where the PE array consists of six PEs using the subject and pattern trees from the example of Figure 6. Each PE cell contains a pattern symbol $E_P(j)$ shown below it in the second row. Depending on the cycle of execution, a PE cell may contain in a vertical fashion from top to bottom: (i) the position of the subject node i, and its Euler chain value $C_S(i)$ grouped inside parentheses, (ii) its occurrence $OS(C_S(i))$ represented by F, L, FL, or N, (iii) its symbol $E_S(i)$ in an italic lower case letter, (iv) the flag bit represented by **1,** and (v) the node positions stored in the stack. The shaded PE is the element that contains a variable. The flag bit represents the match bit that shifts across the array. The positions where the pattern matches the subject are indicated by **Match** in the leftmost PE in the array.

Let n, m and a be the sizes of subject, pattern and array respectively. Each time the pattern is loaded, the subject string is passed through the array once, for processing. The number of times the pattern has to be partitioned, depends on the array size and the pattern length, and is given by $\lceil m/a \rceil$. If the array size is comparable to the pattern length, the performance of the algorithm becomes almost linear. Table 1 provides a general comparison of our hardware algorithm to previously proposed algorithms.

6. Architecture and Chip Layout.

Figure 9, shows the logic diagram of the processing element. The important modules are the 8-bit comparator, 2-bit comparator, 16 x 8 bit stack and the multiplexors. The 8-bit comparator is used to compare the subject and pattern symbols. The inputs to the 8-bit comparator is given from two 8-bit 2x1 multiplexor. The vpe signal, which indicates whether the PE holds a variable or a non-variable, acts as the select signal for the two processors. When the PE holds a non-variable, the subject and pattern symbols, $E_S(i)$ and $E_P(j)$, are compared. When the PE holds a variable, the euler chain value of the subject symbol and the one in stack is compared. The 4x1 multiplexor, selects the appropriate value of the outgoing flag bit, depending on the occurrence status of subject symbol.

Authorized licensed use limited to: New Jersey Institute of Technology. Downloaded on March 24,2010 at 12:27:47 EDT from IEEE Xplore. Restrictions apply.

Iteration 1: Load the pattern substring *fYfaff* and execute the algorithm.

Cycle	FIFO	PE_1	PE_2	PE_3	PE_4	PE_5	PE_6
		F	***Y***	*f*	*a*	*f*	*f*
1			(15,7) FL *a*		(16,3) L *f*		(17,1) L *f* **1**
2		(14,3) N *F*		(15,7) FL *a*		(16,3) L *f* **1**	
3			(14,3) N *f*		(15,7) FL *a* **1**		(16,3) L *f* **1**
4		(13,6) FL *B*		(14,3) N *f* **1**		(15,7) FL *a*	
5			(13,6) FL *b* **1**		(14,3) N *f*		(15,7) FL *a*
6		(12,3) F *f* **1**		(13,6) FL *b*		(14,3) N *f*	
7	**1**		(12,3) F *f*		(13,6) FL *b*		(14,3) N *f*
8		(11,1) N *f*		(12,3) F *f*		(13,6) FL *b*	
9			(11,1) N *f*		(12,3) F *f*		(13,6) FL *b*
10		(10,2) L *f*		(11,1) N *f*		(12,3) F *f*	
11			(10,2) L *f*		(11,1) N *f*		(12,3) F *f*
12		(9,5) L *f*		(10,2) L *f*		(11,1) N *f*	
13			(9,5) L *f*		(10,2) L *f*		(11,1) N *f*
14		(8,9) FL *a*		(9,5) L *f*		(10,2) L *f*	

Authorized licensed use limited to: New Jersey Institute of Technology. Downloaded on March 24,2010 at 12:27:47 EDT from IEEE Xplore. Restrictions apply.

Iteration 1 Continued

Cycle	FIFO	PE_1	PE_2	PE_3	PE_4	PE_5	PE_6
		F	***Y***	*f*	*a*	*f*	*f*
15			(8,9) FL *a*		(9,5) L *f*		(10,2) L *f* **1**
16		(7,5) N *f*		(8,9) FL *a*		(9,5) L *f* **1**	
17			(7,5) N *f*		(8,9) FL *a* **1**		(9,5) L *f* **1**
18		(6,8) FL *a*		(7,5) N *f* **1**		(8,9) FL *a*	
19			(6,8) FL *a* **1**		(7,5) N *f*		(8,9) FL *a*
20		(5,5) F *f* **1**		(6,8) FL *a*		(7,5) N *f*	
21	**1**		(5,5) F *f*		(6,8) FL *a*		(7,5) N *f*
22		(4,2) N *f*		(5,5) F *f*		(6,8) FL *a*	
23			(4,2) N *f*		(5,5) F *f*		(6,8) FL *a*
24		(3,4) FL *b*		(4,2) N *f*		(5,5) F *f*	
25			(3,4) FL *b*		(4,2) N *f*		(5,5) F *f*
26		(2,2) F *f*		(3,4) FL *b*		(4,2) N *f*	
27			(2,2) F *f*		(3,4) FL *b*		(4,2) N *f*
28		(1,1) F *f*		(2,2) F *f*		(3,4) FL *b*	

Authorized licensed use limited to: New Jersey Institute of Technology. Downloaded on March 24,2010 at 12:27:47 EDT from IEEE Xplore. Restrictions apply.

Iteration 2: Load the remaining pattern substring *fXf* and execute the algorithm.

Cycle	PE_4	PE_5	PE_6	FIFO
	f	*X*	*f*	
1	(13,6) FL *b*		(14,3) N *f*	
2		(13,6) FL *b*		
3	(12,3) F *f*		(13,6) FL *b*	
4		(12,3) F *f*		
5	(11,1) N *f*		(12,3) F *f*	
6		(11,1) N *f*		**1**
7	(10,2) L *f*		(11,1) N *f* **1**	
8		(10,2) L *f* **store 2**		
9	(9,5) L *f*	stored 2	(10,2) L *f*	
10		(9,5) L *f* (2)		
11	(8,9) FL *a*	stored 2	(9,5) L *f*	
12		(8,9) FL *a* stored 2		
13	(7,5) N *f*	stored 2	(8,9) FL *a*	
14		(7,5) N *f* stored 2		

Authorized licensed use limited to: New Jersey Institute of Technology. Downloaded on March 24,2010 at 12:27:47 EDT from IEEE Xplore. Restrictions apply.

Cycle	PE_4	PE_5	PE_6	FIFO
	f	***X***	*f*	
15	(6,8) FL *a*	stored 2	(7,5) N *f*	
16		(6,8) FL *a* stored 2		
17	(5,5) F *f*	stored 2	(6,8) FL *a*	
18		(5,5) F *f* stored 2		
19	(4,2) N *f*	stored 2	(5,5) F *f*	
20		(4,2) N *f* stored 2		**1**
21	(3,4) FL *b*	stored 2	(4,2) N *f* **1**	
22		(3,4) FL *b* **1** stored 2		
23	(2,2) F *f* 1 **Match**	stored 2	(3,4) FL *b*	
24		(2,2) F *f* **1**		
25	(1,1) F *f* 1 **Match**		(2,2) F *f*	

Iteration 2 Continued

Authorized licensed use limited to: New Jersey Institute of Technology. Downloaded on March 24,2010 at 12:27:47 EDT from IEEE Xplore. Restrictions apply.

Algorithms		Time Complexity	Number of Processors	Computational Model
Bottom-up	[12]	$O(2^m+n)$	1	Sequential
Top-down	[12]	$O(mn)$	1	Sequential
Kosaraju	[15]	$O(nm^{0.75}\text{polylog}(m))$	1	Sequential
Dubiner et. al	[13]	$O(nm^{0.5}\text{polylog}(m))$	1	Sequential
Ramesh et. al	[1]	$O(m+nk)$	1	Sequential
Ramesh et. al: alg. 1	[17]	$O(n\log n)$	$O(n^{2-\epsilon})$	EREW PRAM
Ramesh et. al: alg. 2	[17]	$O(\log^2 n)$	$O(nk / \log^2 n)$	CREW PRAM
Tarora et al	[18]	$O(\log n)$	$\frac{nm}{\log n}$	CREW PRAM
Smith and Lin	[19]	$O(m+n)$		VLSI
Krishna et. al	[20]	$O(m+n)$	m	VLSI
This algorithm		$O(\lceil m / a \rceil n)$	m	VLSI

n = size of subject, m = size of pattern, k = number of variables in the pattern.

Table 1. Summary of tree pattern matching algorithms.

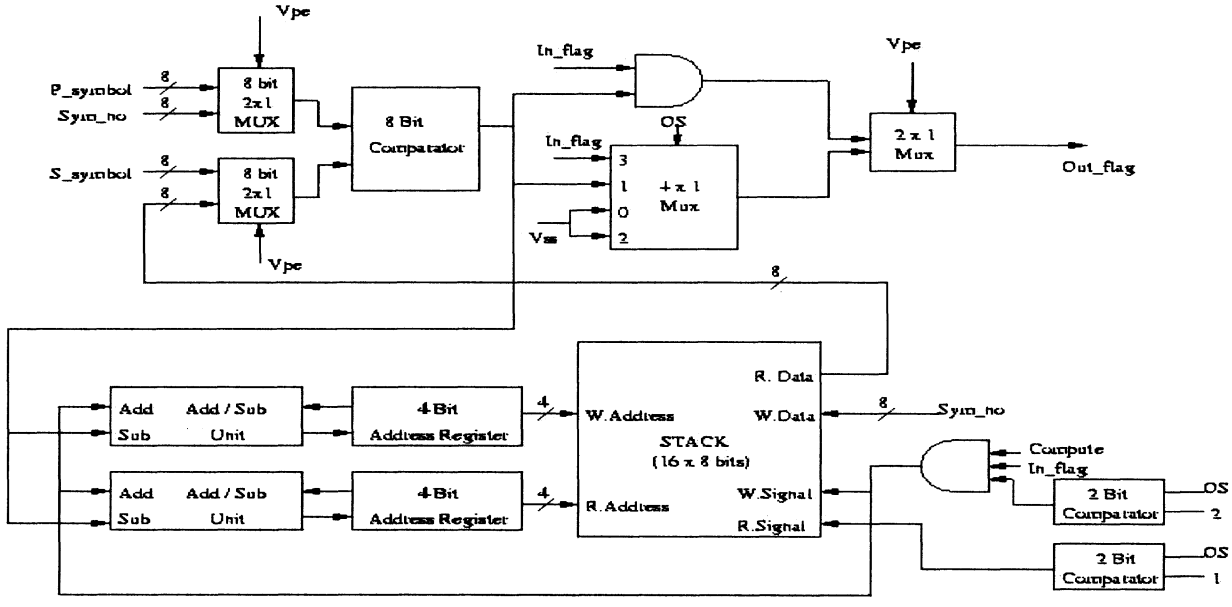


Figure 9. Architecture of PE.

The proposed algorithm was simulated and verified using VHDL at behavioral level. The architecture was then designed at RT-level using VHDL. It was imported into Cadence virtuoso tool and verified using verilog-XL. The HDL design was then synthesized using Cadence design tools, virtuoso and silicon ensemble. The power and delay estimation was done using HSpice. The prototype chip was designed using 0.8 micron technology, with a two phase clock at 333 MHz. The prototype chip consists of four processing elements and a FIFO buffer unit. The final layout of the chip is shown in figure 10. The chip fits into a small frame size of 0.7 mm x 1.08 mm. The small chip area allows us to pack a large number of PEs in to the array.

Authorized licensed use limited to: New Jersey Institute of Technology. Downloaded on March 24,2010 at 12:27:47 EDT from IEEE Xplore. Restrictions apply.

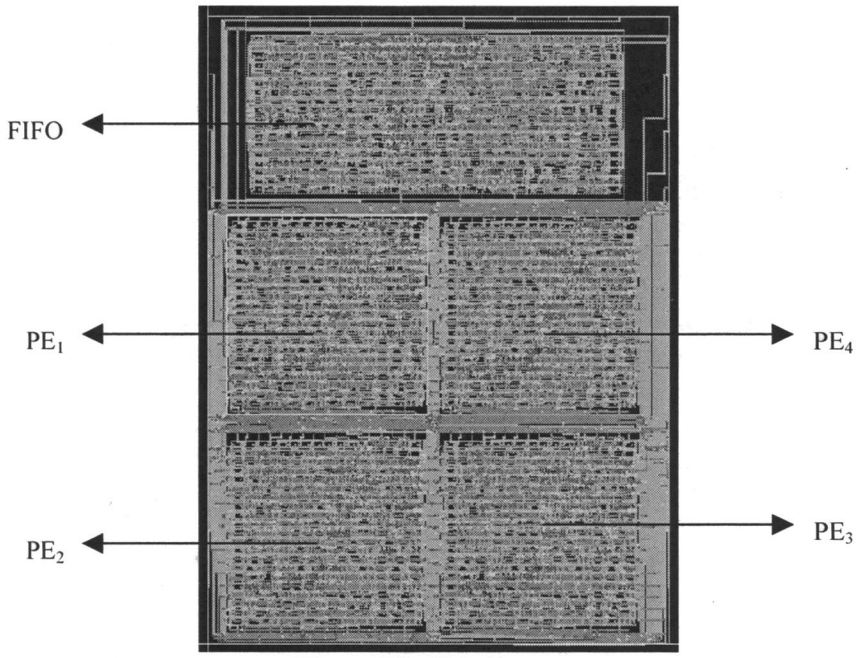


Figure 10. Chip Layout.

In [19], the proposed architecture consists of a set of rows where each row contains a CAM, a shift register, and a horizontal bitstack. In general, CAMs tend to consume substantial silicon area and slow the global clock of the array. The bitstacks were used to contain candidate subtrees for replacement. In [19], trees were represented in parenthesized strings where a left and a right parenthesis represent the start and end of a tree respectively. When a subtree is identified for replacement, a 1-bit is shifted deep into the bitstack for each left parenthesis and shifted out of the stack for each right parenthesis. When the bit emerges out of the stack, an entire subtree has been identified for replacement. However, it is possible that the bitstack may not be deep enough to contain a large subtree. As a result, the partial match may fail. Our architecture does not suffer either from the costs incurred with the extensive use of CAMs, or from the limitation imposed by the fixed capacity of the bitstacks. In [20], the PE array can only handle pattern substrings that fit into the array. In addition, the array can process only one substring at a time. Furthermore when a partial match is identified, a set of match parameters is extracted and then shifted to a CAM PE at the end of the array. The role of this CAM PE is to compare match parameters to each other in order to identify legal attachments. Our architecture is not restricted by either limitation. Moreover, it does not have any CAM PE since it does not shift any heavy data across the PE array. Our architecture is consistently uniform and linear in the sense that all PEs have identical architecture where the only data shifted across the PE array is a single bit.

7. Conclusion

In this paper we presented an efficient parallel hardware algorithm for object recognition using tree pattern matching. Tree pattern matching is an important computational problem encountered in many applications. The algorithm was implemented as a single VLSI chip which is capable of operating at a clock speed of 333 MHz, with running time of $O(\lceil m/a \rceil n)$ where n, m and a are lengths of subject tree, pattern tree and processor array respectively. The algorithm improves on all previous ones in terms of efficiency and flexibility.

0-7803-7970-5/03/$20.00 ©2003 IEEE

Authorized licensed use limited to: New Jersey Institute of Technology. Downloaded on March 24,2010 at 12:27:47 EDT from IEEE Xplore. Restrictions apply.

References

[1] R. Ramesh and I. V. Ramakrishnan, "Nonlinear Pattern Matching in Trees," *Journal of the ACM*, vol. 39, No. 2, pp. 295-316, 1992.

[2] R. Milner, "A Proposal for Standard ML," *Proc. ACM Symposium on LISP and Functional Programming*, 1984, pp. 184-197.

[3] M. J. O'Donnell, "Equational Logic as a Programming Language," in *Foundations of Computer Science*: MIT Press, 1985.

[4] C. M. Hoffman and M. J. O'Donnell, "An Interpreter Generator Using Tree Pattern Matching," *Proc. 6th ACM Symposium on Principles of Programming Languages*, 1979,

[5] A. V. Aho and M. Ganapathy, "Efficient Tree Pattern Matching: An Aid to Code Generation," *Proc. 18th ACM Symposium on Principles of Programming Languages*, 1984,

[6] D. E. Knuth and P. Bendix, "Simple Word Problems in Universal Algebras," in *Computational Problems in Abstract Algebra*. Oxford, England: Pergamon Press, 1970.

[7] J. V. Guttag, E. Horowitz, and D. R. Musser, "Abstract Data Type and Software Validation," *Communications of the ACM*, vol. 21, No. 12, , Dec. 1978.

[8] B. Buchberger and R. Loos, "Algebraic Specification, Computer Algebra and Symbolic Algebraic Computation," in *Computing Supplements*, vol. 4, B. Buchberger, G. E. Collins, and R. Loos, Eds. New York: Springer-Verlag, 1982, pp. 11-43.

[9] R. Nevatia and C. C. Reinhart, "Parallel Processing for Spatial Grouping and Matching," *Proc. 12th International Conference on Pattern Recognition*, Oct. 1994, pp. 290-294.

[10] A. Sanfeliu and M. Ananos, "A CAD Based Vision System for Identifying Industrial Workpieces," *Proc. 6th IFA/IFIP/IFORMS/IMACS Symposium on Information Control Problems in Manufacturing Technology*, 1989,

[11] M. Schaffer and T. Chen, "A VLSI Architecture for 2D Object Classification Based on Tree Matching," *Proc. Computer Architectures for Machine Perception*, Sept. 1995, pp. 138-143.

[12] C. M. Hoffman and M. J. O'Donnell, "Pattern Matching in Trees," *Journal of the ACM*, vol. 29, No. 1, , Jan. 1982.

[13] M. Dubiner, Z. Galil, and E. Magen, "Faster Tree Pattern Matching," *Proc. 31st IEEE Symposium on Foundations of Computer Science*, Oct. 1990, pp. 145-150.

[14] K. Kojima, "A Pattern Matching Algorithm in Trees," in *Lecture Notes in Computer Science*, vol. 147, 1983.

[15] R. S. Kosaraju, "Efficient Tree Pattern Matching," *Proc. 30th IEEE Symposium on Foundations of Computer Science*, Oct. 1989, pp. 178-183.

[16] R. Ramesh, R. M. Verma, T. Krishnapasad, and I. V. Ramakrishnan, "Term Matching on Parallel Computers," *Journal of Logic Programming*, vol. 6, No. 3, , 1989.

[17] R. Ramesh and I. V. Ramakrishnan, "Parallel Tree Matching," *Journal of Symbolic Computation*, vol. 9, No. 4, , 1990.

[18] K. Tarora, T. Hirata, and Y. Inagaki, "A Parallel Algorithm for Tree Pattern Matching," *Systems and Computers in Japan*, vol. 24, No. 5, , 1993.

[19] D. R. Smith and J. C. Lin, "The Tree-Match Chip," *IEEE Transactions on Computers*, vol. 40, No. 5, pp. 250-260, 1991.

[20] V. Krishna, N. Ranganathan, and A. Ejnioui, "A Tree-Matching Chip," *IEEE Transactions on VLSI*, vol. 7, No. 2, pp. 277-280, June 1999.

Authorized licensed use limited to: New Jersey Institute of Technology. Downloaded on March 24,2010 at 12:27:47 EDT from IEEE Xplore. Restrictions apply.