Transactions Briefs

A Tree-Matching Chip

Vamsi Krishna, N. Ranganathan, and Abdel Ejnioui

Abstract—Tree matching is an important problem used for three-dimensional object recognition in image understanding and vision systems. The objective of tree matching is to find the set of nodes at which a pattern tree matches a subject tree. In this paper, we describe the design and implementation of a very large scale integration (VLSI) chip for tree pattern matching. The architecture is based on an iterative algorithm that is mapped to a systolic array computational model and takes $O((n + m)/a)$ time to process a subject of size $n$ using $a$ processors where $a$ is the length of the largest substring in the pattern and $t$ is the number of substrings in the pattern. The variables and nonvariables of the pattern tree are processed separately, which simplifies the hardware in each processing element. The proposed partitioning strategy is independent of the problem size and allows larger strings to be processed based on the array size. A prototype CMOS VLSI chip has been designed using the Cadence design tools and the simulation results indicate that it will operate at 3.33 MHz.

Index Terms—Object recognition, parallel algorithms, systolic, tree matching, VLSI.

I. INTRODUCTION

Tree matching is an important problem in computing used for three-dimensional (3-D) object recognition in image understanding and vision systems. Tree 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. Hoffman and O’Donnell [1] proposed two algorithms: bottom-up and top-down. The bottom-up algorithm has a time complexity $O(2^m + n)$, where $m$ and $n$ are the pattern and the subject sizes, respectively. On the other hand, the top-down algorithm has a much better time complexity of $O((mn)/2 \log(m))$ by reducing the trees into strings. In [2], Kosaraju proposed an algorithm which runs in $O((mn/\log(m)) \log(m))$ and was slightly modified later on by Dubiner et al. [3] to a time complexity of $O((mn)^{0.5} \log(m))$. Polylog refers to any polynomial of $\log$.

Two parallel random-access machine (PRAM) algorithms by Ramesh et al. [4], [5] have been proposed; one runs on a exclusive read exclusive write (CREW) PRAM in $O(n \log n)$ time on $(n^2 - n)$ processors and the other runs on a concurrent read exclusive write (CREW) PRAM in $O((\log^2 n)$ time on $(nk/\log n)$ processors. Recently, Tarora et al. [6] presented a parallel algorithm that runs on the CREW PRAM model using $(nn/\log n)$ processors and runs in $O(\log n)$ time. Smith and Lin described a chip design for tree matching [7]. The architecture relies on the extensive use of content addressable memory (CAM), shift registers, and 1-b-wide stacks for each row of the CAM. Extra logic is introduced to account for replacement of large subtrees in the subject. The additional circuitry requires complex control logic and reduces the regularity of design. Two linear-array hardware algorithms have been proposed in [8]. These algorithms take $O(n + m)$ time to process a subject of size $n$ and a pattern of size $m$ using either $n$ or $m$ processors. The size of the array is dependent on the size of the strings being processed.

II. DEFINITIONS AND TERMINOLOGY

The various definitions and the associated concepts used in the paper are illustrated using an example given in Fig. 1. Let $S$ be the subject tree and $P$ be the pattern tree. The variables in $P$ are represented by squares, while the nonvariables are represented by circles. The variables appear as leaf nodes in the pattern tree. Processing the trees requires the transformation of trees into strings. For this purpose, we number the nodes in the trees in a breadth-first manner. From Fig. 1, we can see that the pattern tree $P$ matches the subject at two nodes: At node 1 of the subject: Variable $X$ replaces the subtree at node 2 and variable $Y$ replaces the subtree at node 6. At node 2 of the subject: $X$ replaces node 4 and $Y$ replaces node 8. We present some definitions which will be used for the processing of the subject and pattern strings.

A. Euler Chain

Denoted by $EC_s$, a Euler chain 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 Fig. 1, the Euler chain of $S$ represented by $(EC_s)$ is 12425859521363731 while the Euler chain of $P$ represented by $(EC_p)$ is 121343531.

B. Euler String

Denoted by $ES_s$, Euler string 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 Fig. 1, the Euler string corresponding to the subject tree represented by $ES_s$, is $f b f f a f a f f f b f a f f f$, while the Euler string corresponding to the pattern tree represented by $ES_p$ is $f X f f Y f a f f f$.

Two trees are identical if their Euler strings are identical. Note that $X$ and $Y$ are variables in the pattern string. Removing those variables

IEEE TRANSACTIONS ON VERY LARGE SCALE INTEGRATION (VLSI) SYSTEMS, VOL. 7, NO. 2, JUNE 1999 277

Publisher Item Identifier S 1063–8210(99)03345-4.
gives substrings denoted by $\sigma_i$. In the above example, we have three substrings, namely, $\sigma_1 = f$, $\sigma_2 = ff$, and $\sigma_3 = faff$. Let the number of variables in the pattern be $k$. The number of substrings is then $(k+1)$.

### C. Occurrence Status

Denoted by $OS$, the occurrence status is a 2-b field associated with each node, indicating first, last, or other occurrence of a tree node in $EC_s$. This can have any of the values zero, one, two, or three, which indicates unknown or middle occurrence, first occurrence, last occurrence, or first and last occurrence, respectively. For example, in Fig. 1, node 1 occurs three times: at position 1, position 11 and position 17. Hence, $OS(1) = 1$, $OS(11) = 0$, and $OS(17) = 2$.

The data structure associated with each node in the subject, and the pattern is shown in Fig. 2. The proposed algorithms use the above data structures to perform the matching.

### III. THE MATCHING ALGORITHM

The algorithm is based on a linear single-instruction multiple-data (SIMD) array architecture with unidirectional systolic data flow. Each PE in the array requires simple hardware components, such as comparators, which are registers to hold information related to the subject and the pattern and control logic. The approach consists of two steps: a preprocessing step, followed by a matching step. The preprocessing step computes the occurrence status $OS$ of each node of the subject. An algorithm to compute the $OS$ is described in [8]. This preprocessing can be interleaved along with the generation of the Euler chain by the host. The preprocessing information for the example of Fig. 1 is given in Fig. 3, where $POS$ is the position of the node in the Euler chain.

The objective is to obtain the set of nodes at which the pattern matches the subject. The partitioning strategy is such that it loads each substring and gets all the partial matches for that substring. For each partition, the subject shifts from the left to right. The substring is loaded in parallel with the subject. Once the substring is loaded and shifted to the rightmost PE, the algorithm is executed. The pattern remains fixed and the subject shifts every clock cycle. The matching process consists of the following three major steps:

1. substring matching;
2. extraction of match parameters (MP’s);
3. testing of legal substitutions and legal attachments.

### A. Substring Matching

This is done for all substrings $\sigma_i$ in the pattern for $1 \leq i \leq k + 1$, where $k$ is the number of variables in the pattern. The three major steps are as follows.

**Step 1:** The first substring $\sigma_1$ has a partial match at position $p$ if and only if: 1) $\sigma_1$ matches $ES$, in each symbol starting from $p$ and 2) $EC_s(p)$ and $EC_s(p + |\sigma_1|)$ are first occurrences.

**Example:** In Fig. 3, $\sigma_1 = f$ has a partial match at position 5 as it satisfies both conditions. It matches $ES = f$ symbol by symbol from position 5 and $OS(5)$ and $OS(6)$ are first occurrences of nodes 5 and 8, respectively.

**Step 2:** $\sigma_i$ where $1 \leq i < k + 1$ has a partial match at some position $p$ of $ES$, if and only if: 1) $\sigma_i$ matches $ES$, in each symbol starting from $p$; 2) $EC_s(p - 1)$ is the last occurrence of some node; and 3) $EC_s(p + |\sigma_i|)$ is the first occurrence of some node.

**Example:** $\sigma_2 = ff$ has a partial match at position 10 as it satisfies the three conditions. It matches $ES = ff$ symbol by symbol from position 10, $OS(9)$ is the last occurrence of node 5 and $OS(12)$ is the first occurrence of node 3.

**Step 3:** $\sigma_{k+1}$ has a partial match at some position $p$ of $ES$, if and only if: 1) $\sigma_{k+1}$ matches $ES$, in each symbol starting from $p$; 2) $EC_s(p - 1)$ and $EC_s(p + |\sigma_{k+1}| - 1)$ are both last occurrences of some nodes.

**Example:** $\sigma_3 = faff$ has a partial match at position 7 as it satisfies the two conditions. It matches $ES = faff$ symbol by symbol from position 7, $OS(6)$ is the last occurrence of node 8 and $OS(10)$ is the last occurrence of node 2. The above steps are mapped to the PE NPE and the computations performed are shown in the following pseudocode:

\[
\text{If pattern symbol = subject symbol} \\
\text{flag1 = 1} \\
\text{If } ES, \text{position }= 1 \text{ then} \\
\text{If } OS < > 1 \text{ then} \\
\text{flag1 = 0} \\
\text{endif} \\
\text{else if } ES, \text{position }= m \text{ then} \\
\text{If } OS < > 2 \text{ then} \\
\text{flag1 = 0} \\
\text{endif} \\
\text{else if } (OS \text{of previous } PE < > 2) \text{ and } (OS \text{of previous } PE < > 3) \text{ then} \\
\text{flag1 = 0} \\
\text{endif} \\
\text{else flag1 = 0} \\
\text{endif}
\]

The algorithm tests for the first symbol of $\sigma_1$ and its first occurrence, the last symbol of $\sigma_{k+1}$ and its last occurrence. $m$ is the number of symbols in the Euler chain of the pattern. The results of all these computations are updated in a flag. If the flag is set, then the
match condition for that particular node is satisfied. The flag of every NPE combines to give the output flag “cflagin.” When this signal is high, it indicates a partial match for a substring has been obtained.

B. Extraction of MP’s

Whenever a partial match for a substring is detected, certain information is stored in a data structure called MP’s. The pattern string is of the form \( \sigma_1 V_1 \sigma_2 V_2 \cdots \sigma_k V_k \), where \( V_1, V_2, \cdots, V_k \) are variables. The computations for obtaining the MP’s are performed by the PE variable processing element (VPE). The data structure of MP is shown in Fig. 4. The fields of the MP’s contain information which is used to detect legal substitutions and attachments and obtain the overall match of the pattern with the subject.

C. Detection of Legal Substitutions and Legal Attachments

Let us assume that the match conditions were satisfied for two consecutive substrings. The variable in the pattern string replaces a C. Detection of Legal Substitutions and Legal Attachments which is used to detect legal substitutions and attachments and obtain MP is shown in Fig. 4. The fields of the MP’s contain information

\[
\text{Fig. 5. Chip architecture.}
\]

Example: The MP’s for \( \sigma_3 = f a f f \) at position 14 is as follows: \( MP_1 = 1, MP_2 = 17, MP_3 = 17, MP_4 = 4, MP_5 = 3 \). Similarly, for \( \sigma_2 = f f \) at position 11, \( MP_1 = 6, MP_2 = 13, MP_3 = 13, MP_4 = 2, MP_5 = 2, \) and for \( \sigma_1 = f \) at position 1, \( MP_1 = 2, MP_2 = 2, MP_3 = 10, MP_4 = 1, MP_5 = 1 \). According to the algorithm, MP’s of \( \sigma_3 \) will be loaded in CPE first. When the MP’s of \( \sigma_2 \) arrive, they are tested for legal attachments. Here, we see that 13 + 4 = 17. Hence, \( \sigma_2 \) legally attaches to \( \sigma_3 \) and variable \( Y \) replaces the node at position 13. The MP’s of \( \sigma_3 \) are now written into CPE and then MP’s of \( \sigma_1 \) arrive. We see that 10 + 2 + 1 = 13 and variable \( X \) replaces the string from position 2 to 10. Hence, \( \sigma_1 \) attaches to \( \sigma_2 \) and the pattern matches the subject at position 2 – 1 = 1.

IV. VLSI DESIGN AND PERFORMANCE

A prototype VLSI chip has been designed and verified using 2-\( \mu \)m CMOS technology. The implementation is based on the systolic dataflow model [9]. The block diagram of the architecture is shown in Fig. 5. The chip is based on a two-phase nonoverlapping clocking scheme and is designed using the Cadence suite of design automation tools. The number of clock cycles to compute the match positions is calculated as follows. Let \( |ES_a| = n, |ES_b| = m \), \# of substrings = \( t \), size of the array \( = a, a \ll (n, m) \). \( t \) cycles are required to load each substring and \( n \) cycles are required for matching since the entire subject has to pass through the array. Hence, the total number of cycles required per substring is \( (a + n) \). Since there are \( t \) substrings, the total processing time is \( T = t(a + n) \). The simulation results indicate that the chip will operate at 33.3 MHz for a 2-\( \mu \)m technology. Its features are shown in Table I. The power consumption is obtained by using the method proposed in [10]. The scaled results for area and power for a 0.8-\( \mu \)m technology using lateral scaling [11] is also shown in the table.

<table>
<thead>
<tr>
<th>Technology</th>
<th>2.0 micron, Scalable CMOS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clocking</td>
<td>2-phase non-overlapping</td>
</tr>
<tr>
<td>IO format</td>
<td>8 bit ascii for subject and pattern</td>
</tr>
<tr>
<td>Pin Details</td>
<td>77 inputs and 9 outputs</td>
</tr>
<tr>
<td>Frame Size</td>
<td>6.9mm x 6.8mm</td>
</tr>
<tr>
<td>Area</td>
<td>NPE: 6.76mm^2, VPE: 1.59mm^2, CPE: 8.67mm^2</td>
</tr>
<tr>
<td>Scaled Area</td>
<td>NPE: 2.7mm^2, VPE: 0.63mm^2, CPE: 3.46mm^2</td>
</tr>
<tr>
<td>Transistors</td>
<td>NPE: 5348, VPE: 2408, CPE: 7306</td>
</tr>
<tr>
<td>Power</td>
<td>93.6mW, Scaled power-234mW</td>
</tr>
</tbody>
</table>

A comparative analysis of the various sequential and parallel algorithms for tree matching is shown in Table II. Among the parallel algorithms, EREW and CREW models have less time complexity, but are theoretical models and are not suitable for implementation. The architecture in [7] is based on memory band and requires complex control logic. The chip implemented in a 2.0-\( \mu \)m 6.8 mm \( \times \) 6.9 mm frame accommodates 120 rows and 45 columns of stack cells. It can perform approximately 300 matchings before an overflow occurs. There is no overflow problem in the architecture proposed in this.
paper since the strings are partitioned and processed based on the array size, thereby simplifying the processing of large strings.

V. CONCLUSIONS

In this paper, we presented an efficient iterative algorithm mapped to a systolic array model. The architecture is simple and has been realized in a single chip. The objective of the prototype is to establish the functioning of the algorithm and hardware. Future work would include designing a board as well as the interface for using the chip in applications such as object recognition and logic optimization.

REFERENCES


An Improved Montgomery’s Algorithm for High-Speed RSA Public-Key Cryptosystem

Chih-Yuang Su, Shih-Arn Hwang, Po-Song Chen, and Cheng-Wen Wu

Abstract—We revise Montgomery’s algorithm such that modular multiplication can be executed two times faster. Each iteration in our algorithm requires only one addition, while that in Montgomery’s requires two additions. We then propose a cellular array to implement modular exponentiation for the Rivest–Shamir–Adleman cryptosystem. It has approximately 2n cells, where n is the word length. The cell contains one full-adder and some controlling logic. The time to calculate a modular exponentiation is about 2n^2 clock cycles. The proposed architecture has a data rate of 100 kb/s for 512-b words and a 100-MHz clock.

Index Terms—Cellular array, data security, modular multiplication, Montgomery’s algorithm, public-key cryptography, RSA.

I. INTRODUCTION

Rivest–Shamir–Adleman (RSA) public key cryptosystem (PKC) is widely used in today’s secure electronic communication. It was invented in 1978 by Rivest et al. [1]. The traditional method to improving data security is based on a common algorithm (or a common key), which is shared by both the transmitter and receiver. A problem of concern is how to send the secret key between the transmitter and receiver. For a PKC, the decryption key is different from the encryption key, and the public key is either of the two keys. To derive the decryption (encryption) key from the public encryption (decryption) key is difficult by definition. If the transmitter encrypts a plaintext by the receiver’s public encryption key, only the receiver has the key to decrypt the ciphertext to its original plaintext. Moreover, the PKC’s also provide a powerful solution to the implementation of digital signature [1], [2]. The security of a PKC is provided by the characteristics of its one-way function. Let f: X → Y denote a one-way mapping, then the calculation of f^{-1}(y) has to be hard, given the calculation of f(x). The famous RSA scheme [1] is based on the Euler and Fermat theorem [3]. Its security is related to the decomposition of N, which is the product of two distinct large prime numbers. It is known that large number decomposition is hard.

The core arithmetic of RSA is modular exponentiation which can be accomplished by a sequence of modular multiplication. Therefore, fast modular multiplication becomes the key to real-time encryption and decryption. An excellent review of the relevant research results on methods for fast modular exponentiation and the description of a very large scale integration (VLSI) processor for modular exponentiation can be found in [4]. However, since the numbers in a usable RSA PKC are very large (512 b or more), its implementation is challenging. Also, due to range comparison and adjustment, the implementation of modular multipliers is much harder than that of normal multipliers [5], [6]. To reduce the time complexity for comparison, a modular multiplication algorithm based on Montgomery’s modular arithmetic

Manuscript received July 31, 1997; revised April 1, 1998 and August 13, 1998. This work was supported in part by the National Science Council, R.O.C., under Contract NSC87-2215-E007-010.

C.-Y. Su, S.-A. Hwang, and C.-W. Wu are with the Department of Electrical Engineering, National Tsing Hua University, Hsinchu 300, Taiwan, R.O.C. P.-S. Chen was with the Department of Electrical Engineering, National Tsing Hua University, Hsinchu 300, Taiwan, R.O.C. He is now with Winbond Electronics Corporation, Hsinchu 300, Taiwan, R.O.C.

Publisher Item Identifier S 1063-8210(99)03344-2.