Bit-Sequential Array for Pattern Matching

NEERAJ TEWARI AND MEGHANAD D. WAGH

This letter presents the design and analysis of a bit-sequential array for pattern matching applications. The architecture makes multiple use of each data sample, has built-in concurrency and pipelining, and is based on a highly modular design with only nearest neighbor connections between array modules. The array computes all occurrences of a pattern of length m, in the string of length n, in O(m + n) time and O(m) hardware. The pattern and the string are fed in sequentially and the match indicators come out in the same fashion, leading to a significant reduction in silicon area.

I. Introduction

Text-editing, visual processing, and signal reconstruction often require searching through a string of characters or bits, looking for instances of a given "pattern" string. The obvious way to search for a matching pattern is to begin searching from the first position in the text and proceeding till a mismatch is found, in which case the starting position is advanced by one. This approach is very inefficient and if all possible matches in the string are to be determined, the worst case time needed is $O(mn)$, where m is the length of the pattern and n is the length of the string.

Researchers have proposed various algorithms which can match the string in $O(m + n)$ time with $O(m)$ hardware (or software) [1]-[4]. In particular, the algorithm by Knuth and Pratt [1] creates a jump table and uses this to implement asynchronous jumps of various steps over the string. Such irregularities in data flow make the hardware implementation of the algorithm inefficient. Other algorithms suffer from similar limitations.

In this letter, a bit-sequential systolic pattern matching architecture is described. Based on a linear array, this structure can find all occurrences of a pattern of length m in a string of length n, in $O(m + n)$ time and $O(m)$ hardware complexity. The characters in both the pattern and the string are assumed to be bits here but the hardware can be easily replicated for parallel matching in case the characters are words and the same time efficiency is desired.

Let the pattern P, the string S, and the output flag vector R be, respectively, represented as

$$P = [a_0 a_1 \cdots a_{m-1}]$$

$$S = [x_0 x_1 \cdots x_{n-1}]$$

and

$$R = [r_0 r_1 \cdots r_{m+n-2}]. \tag{1}$$

As explained earlier, each component of P and S is either 0 or 1. The architecture sets the output bit r_j to 1 if the substring $x_{j-m+1}x_{j-m+2} \cdots x_j$ matches the pattern P and 0 otherwise. Thus letting $\wedge$ denote the AND operation

$$r_j = [(x_{j-m+1} = a_0) \wedge (x_{j-m+2} = a_1) \cdots \wedge (x_j = a_{m-1})]. \tag{2}$$

r_j values for $j = 0, 1, \cdots, m-2$ are ignored and other r_j's are renamed as $r_j = F(j - m + 1)$, $(m-1) \le j \le (m + n - 2)$. Without loss of generality we can assume $m = 2M$ and express (2) as

$$\begin{aligned} F(j) &= \prod_{i=0}^{2M-1} \neg \{a(i) \oplus x(j+i)\} \\ &= \prod_{i=0}^{2M-1} f[a(i), x(j+i)] \quad \text{(say)} \end{aligned} \tag{3}$$

where $\oplus$ stands for exclusive-or, $\neg$ is the logical *not*, and Π is a multioperand AND. Architecture to evaluate (3) is presented in the next section.

II. Bit-Sequential Linear Array Architecture

Let the input bit-string $\{x_j\}$ be fed into the architecture bit-serially, with x_0 first and the output bit-string $\{r_j\}$ be extracted bit-serially, with r_0 first. At this point we assume tht the pattern string $\{a_j\}$ is already loaded in the architecture and does not move as do the $\{x_j\}$ and $\{r_j\}$ strings. Later we will show that even $\{a_j\}$ could be loaded bit-serially without compromising on the time efficiency. The advantages of such "Bit-Sequential" architectures are that they can be pipelined to the bit level permitting a very high clocking rate, they use repeated modules with nearest neighbor intermodular communication paths of width 1, and they have the fewest possible input and output pins. These characteristics make these architectures ideal for Very Large Scale Integration (VLSI) implementation [5], [6].

Let Z denote the delay operator associated with data input rate. Then (3) may be transformed into

$$\begin{aligned} F(j) = &\prod_{i=0}^{M-1} \{f[a(2i), x(j+2i)]\} \\ &\wedge \prod_{i=0}^{M-1} \{f[a(2i+1), x(j+2i+1)]\}. \end{aligned}$$

Since the pattern string $\{a_j\}$ is time-invariant, i.e., it does not flow in the array with time, the delay operator can be distributed as

$$\begin{aligned} = &\prod_{i=0}^{M-1} \{Z^{-i}(f[a(2i), Z^{-i}x(j)])\} \\ &\wedge Z^{-1} \prod_{i=0}^{M-1} \{Z^{-i}(f[a(2i+1), Z^{-i}x(j)])\}. \end{aligned}$$

This expression shows that each flag $F(j)$ is computed by ANDing outputs of two multioperand ANDs. The ith operand for each multioperand AND is obtained, in turn, by delaying by i time units the result of the comparison of $a(2i)$ (or $a(2i+1)$) and $x(j)$ delayed by i time units. This immediately suggests the architecture shown in Fig. 1. We now show that the output of this architecture provides us the desired flag vector.

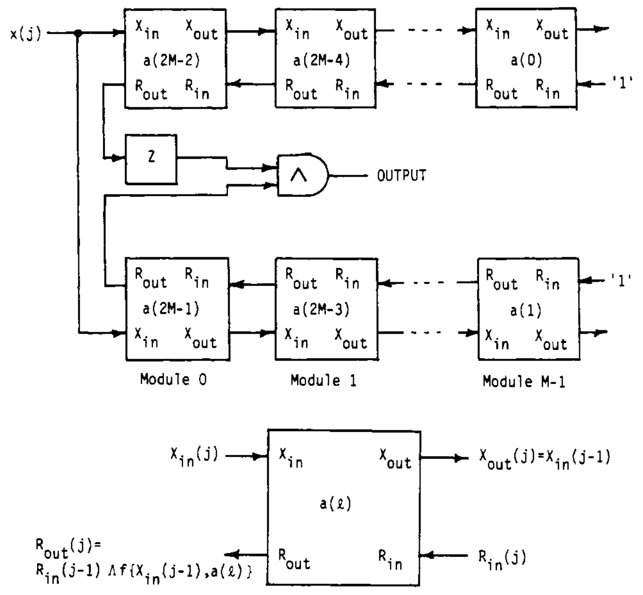

Fig. 1. The proposed bit-sequential architecture for pattern matching, showing the constants stored in various modules and the functionality of each module at time j.

Manuscript received March 31, 1986. This work was partially supported by AT&T Information Systems.

The authors are with the Computer Science and Electrical Engineering Department, Lehigh University, Bethlehem, PA 18015-3084, USA.

0018-9219/86/1000-1465$01.00 © 1986 IEEE

Theorem: The output of this architecture $O(j)$ at time j is related to the flags as

$$O(j + 2M) = F(j).$$

Proof: Let the ith module of the upper bank of Fig. 1 hold $a(2(M - i) - 2)$ and the lower one, $a(2(M - i) - 1)$. Further, let

$$\text{VAR}\ (i, j, k), \quad 0 \le i \le M - 1, \quad j \ge 0, \quad k = \text{upper or lower}$$

be the value of the variable VAR for the ith module at the jth clock. Index k associates the variable with the upper or the lower array. The geometry of the architecture then yields

$$\begin{aligned} X_{in}(i, j, \text{upper}) &= X_{in}(i, j, \text{lower}) = x(j - i) \\ R_{out}(i, j + 1, \text{upper}) &= R_{out}(i + 1, j, \text{upper}) \\ &\quad \wedge \{ f[a(2(M - i) - 2), X_{in}(i, j, \text{upper})]\} \\ &= R_{out}(i + 1, j, \text{upper}) \\ &\quad \wedge \{ f[a(2(M - i) - 2), x(j - i)]\} \end{aligned}$$

and similarly,

$$\begin{aligned} R_{out}(i, j + 1, \text{lower}) &= R_{out}(i + 1, j, \text{lower}) \\ &\quad \wedge \{ f[a(2(M - i) - 1), x(j - i)]\}. \end{aligned} \tag{4}$$

We now have

$$\begin{aligned} O(j + 2M) &= R_{out}(0, j + 2M - 1, \text{upper}) \\ &\quad \wedge R_{out}(0, j + 2M, \text{lower}). \end{aligned} \tag{5}$$

Using recursion on (4)

$$\begin{aligned} R_{out}(0, j + 2M - 1, \text{upper}) &= R_{in}(M - 1, j + M - 2, \text{upper}) \\ &\quad \wedge \prod_{i=0}^{M-1} \{ f[a(2i), x(k + 2i)]\} \\ &= \prod_{i=0}^{M-1} \{ f[a(2i), x(k + 2i)]\} \end{aligned} \tag{6}$$

and

$$\begin{aligned} R_{out}(0, j + 2M, \text{lower}) &= R_{in}(M - 1, j + M - 1, \text{lower}) \\ &\quad \wedge \prod_{i=0}^{M-1} \{ f[a(2i + 1), x(k + 2i + 1)]\} \\ &= \prod_{i=0}^{M-1} \{ f[a(2i + 1), x(k + 2i + 1)]\}. \end{aligned} \tag{7}$$

Finally (5)-(7) together yield

$$\begin{aligned} O(j + 2M) &= \prod_{i=0}^{M-1} \{ f[a(2i), x(j + 2i)]\} \\ &\quad \wedge \prod_{i=0}^{M-1} \{ f[a(2i + 1), x(j + 2i + 1)]\} \\ &= \prod_{i=0}^{2M-1} \{ f[a(i), x(j + i)]\} \\ &= F(j). \end{aligned}$$

Q.E.D.

III. Discussion and Conclusions

The pattern matching architecture proposed here can be very easily implemented. The function $f[a, b]$ is built as $\neg(a \oplus b)$, where $\neg$ represents a NOT and $\oplus$ an EXCLUSIVE OR. Each module will contain this logic to compute f besides three flip-flops (delays) to hold a component of a (stationary), a component of x (moving to the right), and a component of the flag (moving to the left). The architecture can thus be clocked at speeds determined by the delay in the computation of f and the flip-flop hold and set up times. Thus the clocking speed is independent of the size of the array.

The concept of "wildcards" (pattern characters that match with everything) can be incorporated in this architecture by using a 2-bit representation for each pattern component and modifying function f as

$$\begin{aligned} f[a, b] &= 1, \quad \text{if } a = b \text{ or if } a \text{ is a wildcard} \\ &= 0, \quad \text{otherwise.} \end{aligned}$$

It was assumed earlier that the a-sequence is stationary and is already resident in the architecture when $x(0)$ comes in. But this is not necessary. If one provides an additional path between modules, the a-coefficients may be fed serially (even a-components to the upper bank and odd to the lower) along with the first M components of x. This is an ideal situation because it does not affect the output delay (the first M clock periods cannot, even otherwise, be used to compute any useful flag component) but gets rid of any loading time for the array. Thus *all* the matches between an m-length pattern and an n-length string can be obtained in this $O(m)$ hardware in $O(m + n)$ time. Control of this architecture is very simple since no flip-flops need to be cleared or preset. The modularity, nearest neighbor connections, data buses of width 1, and the simple control makes this architecture suitable for *VLSI* implementation.

References

[1] D. E. Knuth, J. H. Morris, and V. R. Pratt, "Fast pattern matching in strings," *SIAM J. Comput.*, vol. 6, no. 2, pp. 323-350, June 1977.
[2] M. J. Foster and H. T. Kung, "The design of special purpose VLSI chips," *Computer*, vol. 13, no. 1, pp. 26-40, Jan. 1980.
[3] M. J. Fischer and M. S. Paterson, "String matching and other products," Mass. Inst. Technol., Product MAC, Tech. Rep. 41, 1974.
[4] R. S. Boyer and J. S. Moore, "A fast string search algorithm," *Commun. ACM*, vol. 20, no. 10, p. 762, Oct. 1977.
[5] N. Tewari and M. D. Wagh, "Bit-sequential signal processing architectures for VLSI implementation," Comput. Arch. Lab., Dept. Comput. Sci. Elec. Eng., Lehigh Univ., Rep. LU-CAL-85-03, Aug. 1985.
[6] ——, "Bit-sequential systolic array filters," Comput. Arch. Lab., Dept. Comput. Sci. Elec. Eng., Lehigh Univ., Rep. LU-CAL-85-05, Dec. 1985.

Near-Optimal Controllers for Synchronous Machines with Prescribed Degree of Stability Via Iterative Separation of Time Scales

N. P. SINGH, Y. P. SINGH, AND S. I. AHSON

Near-optimal controller for synchronous machines with prescribed degree of stability using the slow subsystem derived through iterative approach is proposed. The closed-loop system poles are constrained to the left of Re (s) < $-\alpha$ where $\alpha > 0$ and $\alpha \in (0, \alpha_0)$. The results are compared with those obtained via the classical quasi-steady-state (qss) technique.

I. Introduction

The design of optimal control system is often confronted with high dimensionality and stiff numerical problems due to the presence of parasitic parameters. To alleviate these difficulties, generally, suboptimal/near-optimal controllers are designed using the reduced-order models of such systems. In this letter we present a procedure for the design of near-optimal controllers for synchronous machines with prescribed degree of stability based on the

Manuscript received April 21, 1986; revised June 16, 1986
The authors are with the Department of Electrical Engineering, Indian Institute of Technology Delhi, New Delhi—110016, India.

0018-9219/86/1000-1466$01.00 © 1986 IEEE