A Hybrid Pipelined Path-Searching Architecture for Multiple Communications Applications

Horng-Dar Lin and David G. Messerschmitt

Abstract—**Many communications applications require similar processing functionality but are implemented independently. In particular, a number of applications (including trellis coding, encryption, and speech recognition) use techniques based on shortest path search algorithms. In this paper, we propose a high-throughput architecture that can search for the shortest path within a graph. The architecture can decode any data encoded with a finite state machine (FSM) or data encrypted in a dynamic trellis code and also serve as a specialized processor for other searching and matching applications. Balance between flexibility and hardware efficiency is achieved by an integrated design of architecture, in-place scheduling, and concurrent algorithms.**

I. Introduction

VARIOUS decoding, speech recognition, pattern matching, and tree searching applications can often be modeled as finding a min-cost path within a (structured) graph using dynamic programming. The graph can be a structured trellis of a finite state machine (FSM), such as in decoding convolutional or trellis codes with a dynamic programming algorithm like the Viterbi algorithm, the stack algorithm, or the M-algorithm [1]. Stereo vision or dynamic-time-warp speech recognition uses similar dynamic programming algorithms to search through a less structured graph. Some previous processors, such as [2] and [3], provide retargetable functionality but their applications are more limited because of their throughput, input/output (I/O) bandwidth, and flexibility. We will describe a new multipurpose engine that uses hybrid pipelined architecture (mixed between serial and parallel pipelined architectures) and specific scheduling to optimize performance and flexibility in decoding and other applications.

II. Architecture

As shown in Fig. 1, the multipurpose architecture consists of a reconfigurable network of piplined processing elements (PPE) and a pipelined shifting buffer operating according to a preprogrammed control sequence or schedule. The reconfigurable routing network can be implemented as a pipelined multiport memory addressed according to the control map or a physical routing network based on combinations of multiplexers and demultiplexers, switched multiple buses, time division multiplexed buses, multistage switches, or crossbar switches. The choice usually depends more on costs and the required flexibility than on the routing delay, since the effects of routing delay can be minimized with pipelining. The PPE illustrated in Fig. 2 looks like a pipelined version of a conventional path-selection processing element except that the selected accumulated metric value is inserted in

Paper approved by C.-P. Jeremy Tzeng, the Editor for VLSI in Communications of the IEEE Communciations Society. Manuscript received August 13, 1992; revised April 21, 1993.

H.-D. Lin is with HolonTech Corp., San Jose, CA 95124 USA.

D. G. Messerschmitt is with the University of California, Department of EECS, Berkeley, CA 94709 USA.

Publisher Item Identifier S 0090-6778(96)09022-8.

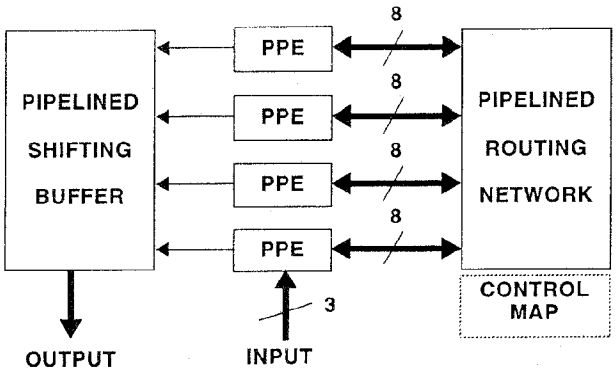


Fig. 1. The functional blocks of the hybrid pipelined processor.

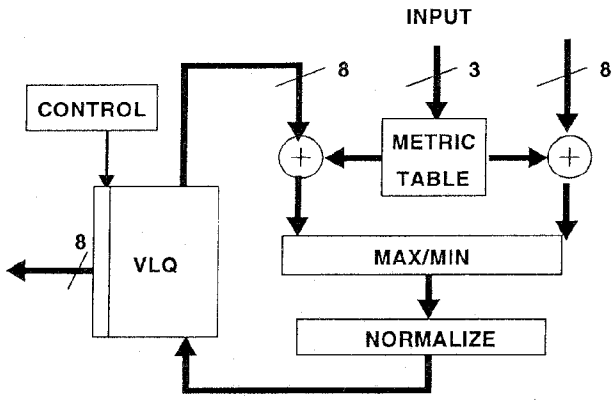


Fig. 2. The functional blocks of the pipelined processing element (PPE). Pipeline latches are not shown here.

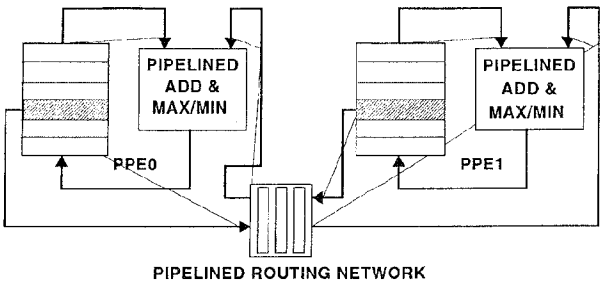
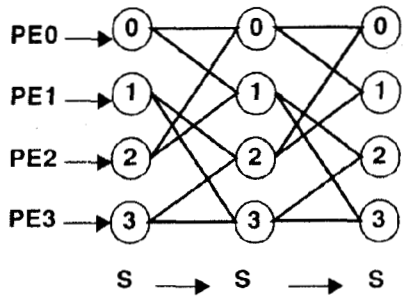
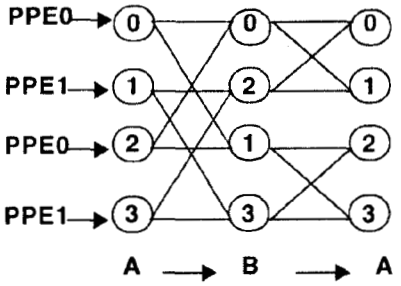


Fig. 3. An example illustrating how to use variable length queue (VLQ) to offset the routing delay.



PE	Metrics computed for			
PE0	S0	S0	S0	S0
PE1	S1	S1	S1	S1
PE2	S2	S2	S2	S2
PE3	S3	S3	S3	S3

(a)



PPE	Metrics computed for			
PPE0	B0	B1	A0	A2
PPE1	B2	B3	A1	A3

A → B B → A

(b)

PPE	DATA PORTS	METRIC VALUES READ OR WRITTEN PER CLOCK CYLE															
PPE0	LOCAL VLQ	A0	+	+	A2	-	-	B0	/	/	B1	*	*				
	READ PORT	A2	+	+	A0	-	-	B2	/	/	B3	*	*				
	WRITE PORT				B0	+	+	B1	-	-	A0	/	/	A2	*	*
PPE1	LOCAL VLQ	A1	+	+	A3	-	-	B2	/	/	B3	*	*				
	READ PORT	A3	+	+	A1	-	-	B0	/	/	B1	*	*				
	WRITE PORT				B2	+	+	B3	-	-	A1	/	/	A3	*	*

(c)

Fig. 4. Example for deriving a two-stage cyclic in-place schedule for the multipurpose architecture. The metric label uses A and B to denote even or odd stage and a state number to denote the associated state. The steps are (a) the orignial trellis and the conventional parallel processing element (PE) schedule, (b) the reordered trellis and the in-place PPE schedule, and (c) the pipelined in-place schedule mapped to the multipurpose architecture.

a variable length queue (VLQ), which is just a group of controllable pipeline latches. The queue length of the VLQ is specified during configuring (programming) the architecture and is usually fixed during processing. The VLQ serves two purposes. First, it provides a convenient swap space for concurrent processing on the PPE. Second, it offsets the routing delay by matching the number of pipeline stages in the pipelined routing network. For example, suppose the pipelined routing network has J pipeline stages. To unskew the routing delay, we put the PPE output to the pipelined routing network at the $(J + 1)$ position from the top of the queue, as illustrated in Fig. 3. The hashed pipeline latches within the VLQ's mark the metric output positions of the PPE's. VLQ's help remove architectural dependency, since the amount of concurrency of the algorithms, the number of pipeline latches within PPE's, and the (pipelined) routing networks can be designed independently and then integrated with the help of VLQ's. Note that the length of the VLQ only affects the processing delay but not the processing throughput of the PPE.

The pipelined shifting buffer in Fig. 1 records the decisions of the max/min units of the PPE's and updates the path records. It is optional because the host processor can process the PPE outputs directly.

III. Deriving Pipelined In-Place Schedule for Applications

To run different applications on the multipurpose architecture, we need to derive a control sequence for the PPE's based on the application and the hardware properties. The local feedback through the VLQ allows the PPE to use in-place scheduling [4], [5] to reduce the metric I/O traffic. In-place scheduling is a way of assigning path selections to (parallel) processors such that each processor always uses locally generated path metrics in the next stage of path selections. General in-place scheduling for rate-k/n convolutional codes, Ungerboeck's codes, and general trellis codes has been solved in a companion paper [5]. Here, we only restate from [5] that in-place scheduling minimizes the intercommunications

between PPE's, and thus allows our architecture to use VLQ's to simplify routing and to reduce communication bandwidth between PPE's.

For example, assume that we use a two-PPE hybrid pipelined processor to run the Viterbi algorithm on the four-state trellis of a convolutional code. The PPE has three pipeline stages in the (adder)-(max/min)-(normalize) data path. The pipelined routing network is a switching network or memory with three pipeline stages. Thus, the PPE takes three clock cycles to compute data; routing data from one PPE to another takes three clock cycles; routing data back to the same PPE (bypassing the routing network) is instantaneous.

The first step is to apply in-place scheduling to reorganize the trellis from the conventional form in Fig. 4(a) into the cyclic form in Fig. 4(b). The resulting PPE in-place schedule is also shown in Fig. 4(b), which has not yet considered concurrency, timing, and hardware constraints. A complete cyclic PPE schedule can be derived easily by expanding Fig. 4(b) according to data dependency and hardware constraints. The cyclic PPE schedule in Fig. 4(c) satisfies both the routing and the PPE clocking constraints: the delay from the PPE's data path inputs ["local VLQ" and "read port" in Fig. 4(c)] to data path's output ("write port") is three clock cycles, and the delay from sending data from a PPE's "write port" to its own or another PPE's "read port" is zero or three cycles, respectively. The VLQ length is three because the delay from an PPE's "write port" to its own "local VLQ" is three cycles. To match the three-cycle routing delay, the PPE's output to the routing network is at the insertion point of the VLQ (the "write port"). It is obvious that we can overlay another two identical cyclic PPE schedules to the one in Fig. 4(c), meaning that we can decode three blocks of codewords concurrently using various concurrent methods (low overhead methods, such as [6]–[9], are more desirable). Similar design works if we scale all hardware parameters accordingly (for example, from three-cycle delay to four-cycle delay with four-block concurrent processing), or if we change the VLQ's "read port" or "write port" to accommodate a different routing delay or pipeline depth.

The multipurpose architecture can process a set of codes called *dynamic trellis codes,* which provide certain degree of communication security. Dynamic trellis codes are generated by nonstationary finite state machines. Nonstationarity implies that the number of states, the state transition function, and the output function can all vary with time. This makes data-interception difficult or infeasible. Our architecture can decode such dynamic trellis codes concurrently with the maximum-likelihood or suboptimal algorithms.

The proposed architecture in general can decode any codes that are encoded with finite state machines. The architecture adapts to different algorithms, codes, and data rates by modifying the control map, the VLQ's, and the metric tables. By merging and splicing control schedule, the multipurpose architecture can simultaneously decode multiple codes. It can be used in generic search problems such as finding the minimum-cost traveling path between any two cities[1] if a map and traveling costs of individual routes are provided.

IV. Conclusion

A flexible architecture is proposed for high throughput decoding for different encoding sources, deciphering encrypted data, voice command recognition and other path searching applications. In-place scheduling and concurrent path searching provide hardware advantages. The architecture can be reused efficiently, saving the cost of duplicated hardware for different services.

References

[1] S. Lin and D. J. Costello, Jr., *Error Control Coding: Fundamentals and Applications.* Englewood Cliffs, NJ: Prentice-Hall, 1983.
[2] S. C. Glinski *et al.*, "The graph search machine (GSM): A VLSI architecture for connected speech recognition and other applications," *Proc. IEEE,* vol. 75, no. 9, pp. 1172–1184, Sept. 1987.
[3] H. Lou and J. M. Cioffi, "A programmable parallel processor architecture for Viterbi detection," in *Proc. GlobeCom'90,* Dec. 1990.
[4] C. M. Rader, "Memory management in a Viterbi decoder," *IEEE Trans. Commun.,* vol. COM-29, pp. 1399–1401, Sept. 1981.
[5] H.-D. Lin and C. B. Shung, "General in-place scheduling for the Viterbi algorithm," in *Proc. ICASSP,* Toronto, Canada, May 14–17, 1991.
[6] T. Fujino *et al.*, "A 120 Mbits/s coded SPSK modem with soft-decision Viterbi decoder," in *Proc. ICC,* Toronto, Canada, June 22–25, 1986.
[7] G. Fettweis and H. Meyr, "A modular variable speed Viterbi decoding implementation for high data rates," in *Proc. EUSIPCO-88,* Sept. 1988.
[8] H.-D. Lin and D. G. Messerschmitt, "Algorithms and architectures for concurrent Viterbi decoding," in *Proc. ICC,* Boston, MA, June 11–14, 1989.
[9] P. Black and T. H.-Y. Meng, "A hardware efficient parallel Viterbi algorithm," in *Proc. ICASSP'90* Apr. 1990.
[10] H.-D. Lin, "Concurrency in trellis searching and traversing algorithms," Ph.D. thesis, Department of EECS, University of California at Berkeley, May 1991.
[11] G. Fettweis and H. Meyr, "Parallel Viterbi algorithm implementation: breaking the ACS-bottleneck," *IEEE Trans. Commun.,* vol. 37, no. 8, Aug. 1989.
[12] H. K. Thapar and J. M. Cioffi, "A block processing method for designing high-speed Viterbi detectors," in *Proc. ICC,* Boston, MA, June 11–14, 1989.

[1] This is not a traveling salesman problem.