A Multiprocessor Architecture for Viterbi Decoders with Linear Speedup

Gennady Feygin, Patrick G. Gulak, *Member, IEEE*, and Paul Chow, *Member, IEEE*

***Abstract*—A family of multiprocessor architectures implementing the Viterbi algorithm is presented. The family of architectures is shown to be capable of achieving an increase in throughput that is directly proportional to the number of processors when the number of processors is smaller than the constraint length v of the code. The hardware utilization and the depth of the pipelining available inside each processor are also shown. An architecture with (v − 1) processors is found to be particularly advantageous, since it results in the maximum speedup and the simplest interconnection structure.**

I. Introduction

With the growing use of digital communications, there has been an increased interest in advanced coding techniques that yield higher coding gain and permit transmission of larger quantities of data over the same channel. As throughput requirements increase, decoders must be able to operate at higher clock speeds. An alternative to higher clock speeds is to utilize the parallelism inherent in the decoding algorithm by splitting the computations between multiple processors, with each processor performing a part of the ensemble of computations. Much research effort has been dedicated to developing array processors for digital signal processing (DSP) applications, including systolic arrays, wavefront arrays, and data-flow arrays.

Recently there has been increased interest in multiprocessor implementations of decoders for the class of error-correction codes known as convolutional codes, and in particular, in a decoding technique known as the Viterbi algorithm [1]. An introduction to the Viterbi algorithm can be found in [2]. The Viterbi algorithm has found widespread acceptance in deep-space communication networks [3]–[5], magnetic disk memory [6], and adaptive channel equalization [7].

A number of multiprocessor architectures have been proposed for Viterbi decoders, including one- and two-dimensional systolic meshes [8], [9] and the perfect shuffle layout [10]. All of these architectures provide a throughput increase that is sublinear (i.e., to increase the throughput by a constant factor more than a constant growth in area is required [19]). An architecture that is closely related to the perfect shuffle, known as the crenellated-FFT [11], is currently being implemented at the Jet Propulsion Laboratory. Yet another family of architectures for Viterbi decoding has been introduced recently by [12]–[14], and these architectures are capable of providing throughput increase that is linear in the number of processors. Unfortunately, throughput increase is also inversely proportional to the number of states in the trellis [14]. This architecture is of benefit only in the case of the short-constraint length codes, since the number of states in the trellis grows exponentially with the constraint length of the code.

Recent work in [9] has been able to take advantage of the relationship between the Viterbi algorithm and the fast Fourier Transform (FFT) [7] by proposing a novel "cascade" Viterbi decoder that is closely related to the "cascade" design for the pipelined FFT computation [15], but requires an additional "recirculation" connection from the output of the last processor to the inputs of the first processor. While the cascade Viterbi decoder architecture has a topology that is regular and requires only local interprocessor communications, making it suitable for VLSI implementation, there is one major drawback—the utilization of the processors is only 50%. This paper proposes an architecture that resolves this shortcoming and provides additional benefits, including simpler switching structure and increased pipelining availability inside the processor units.

II. Organization of this Paper

We begin in Section III with a discussion of the mapping and scheduling of the operations in the dependence graph of the Viterbi algorithm (trellis diagram) that are characteristic of all members of the generalized cascade Viterbi decoders (GCVD) family of architectures. In Section IV we analyze the flow of computations in the "cascade" Viterbi decoder of [9] (we will use the name canonic cascade Viterbi decoder (CCVD) for this architecture).

In Section V we introduce an alternative architecture, a folded cascade Viterbi decoder (FCVD) architecture that uses half as many processors, but achieves a throughput equal to that of CCVD. This is followed, in Section VI, by a formal algebraic proof of the existence of the GCVD

Manuscript received December 3, 1990; revised October 12, 1992. The associate editor coordinating the review of this paper and approving it for publication was Dr. John Eldon. This work was supported by the Natural Sciences and Engineering Research Council of Canada and by the Information Technology Research Center of Ontario.

The authors are with the Department of Electrical and Computer Engineering, University of Toronto, Toronto, Ontario, Canada M5S 1A4.

IEEE Log Number 9210108.

family of architectures, which leads to an interesting result about the availability of pipelining in the GCVD family of architectures.

In Section VII we derive an expression for the utilization of the processors in a GCVD as a function of the number of processors k.

Section VIII contains an analysis of the cross-point switches and interstage registers that are required to ensure correct spatial and temporal alignment of the path metrics at the inputs of each processor. Section IX discusses survivor sequence memory management for GCVD. We also compare our proposed architecture with uniprocessor and fully parallel Viterbi decoders. Finally, Section X summarizes the main results of this paper.

III. The Trellis Diagram: Mapping and Scheduling of Operations

Consider a portion of the trellis diagram for the (2, 1, 4) convolutional code illustrated in Fig. 1. The nodes in the trellis diagram correspond to the add-compare-select (ACS) computations and the edges indicate dependencies between the ACS computations of the successive stages of the trellis. Given a number of individual ACS units we can derive a large number of possible assignments of the nodes of the trellis diagram to the physical ACS units. In this paper we will concentrate on a particular assignment: all computations in a given stage of the trellis are performed on a single ACS unit. We must also select a schedule of computations in each ACS unit that does not violate causality. If more than one such schedule can be found, we then choose one schedule that maximizes the utilization of the hardware. The solid-lines in Fig. 1 give an indication of the scheduling constraints imposed by the causality requirement. For instance, to perform the first ACS computation of the last (rightmost) stage, two ACS computations must be performed in the previous stage, four in the one before that; a total of $N = 16$ computations in the leftmost stage must be performed. Note that Fig. 1 indicates two ACS computations being performed simultaneously in any stage of the trellis, in a manner reminiscent of the FFT butterfly computation. Since two path metrics (say pm_{2i} and pm_{2i+1}) are used to compute two path metrics (pm_i and $pm_{i+N/2}$) of the next stage and no other path metrics, it is advantageous to perform ACS operations two at a time.

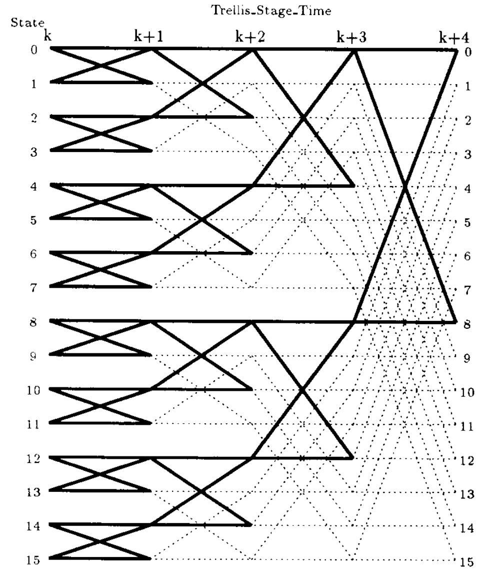


Fig. 1. The dependence graph of the ACS operations over $\nu - 1$ stages of the trellis.

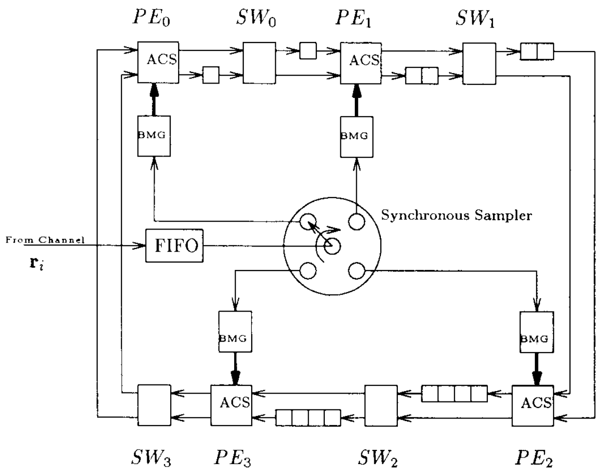


Fig. 2. An example of the canonic cascade Viterbi decoder with binary alphabet and constraint length 4. ACS: Add-compare-select section of the processor. BMG: Branch metric generator. The survivor sequence management is not shown.

IV. The Canonic Cascade Architecture

The Canonic cascade architecture of the Viterbi decoder was originally introduced in [9]. In the CCVD architecture log (N) processors are arranged in a ring, together with local memory and switches. An illustration of the CCVD decoder for $q = 2$ (binary input alphabet), $\nu = 4$, $N = q^{\nu} = 16$ is given in Fig. 2. With the exception of the feedback path connection and the input branch generation circuitry, the architecture of the CCVD is identical to that used for bitonic sorting [16] and the FFT [15]. The canonic cascade layout is regular and compact, and has only local (nearest neighbor) communication so that partitioning the decoder into multiple chips (or boards) is straightforward. The CCVD layout can be easily extended to accommodate any problem size by inserting more processors, memory, and switches in the ring structure.

There are four types of circuits required to implement the state information update section of the CCVD, as illustrated in Fig. 2. In the binary input alphabet case each processor (PE_j, $0 \leq j \leq \nu - 1$) consists of two add-compare-select (ACS) circuits in a butterfly configura-

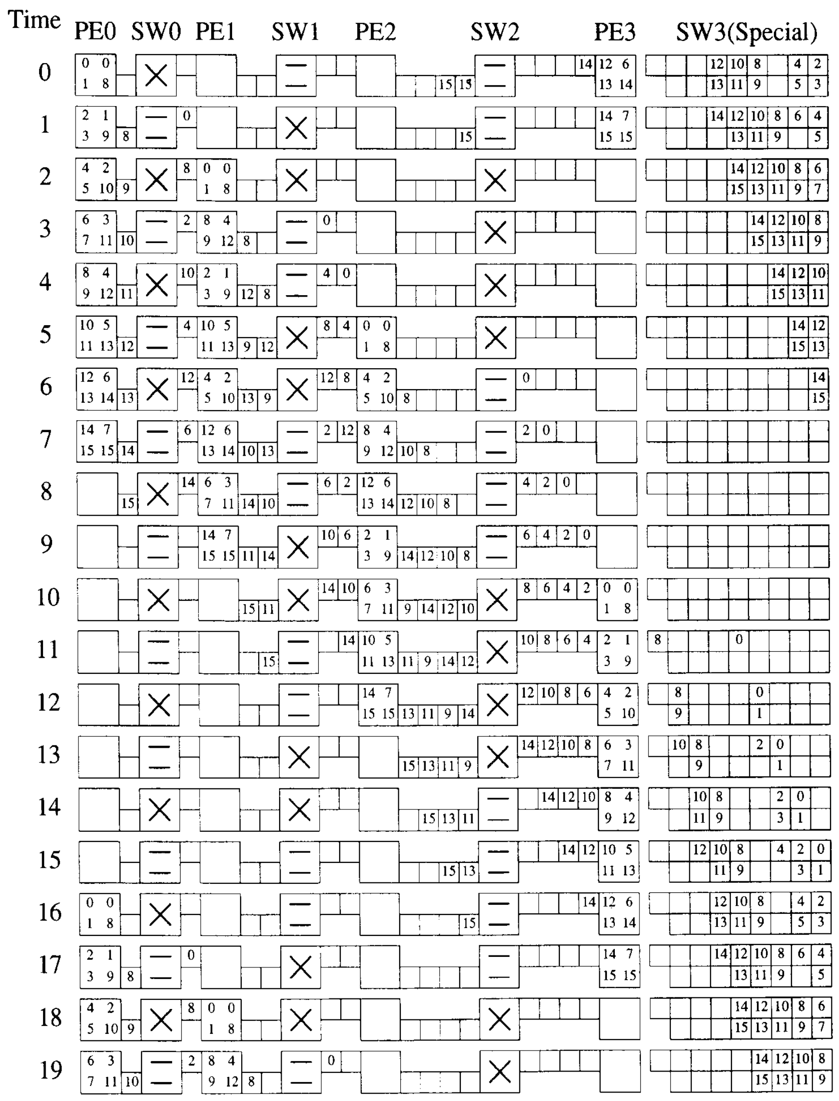


Fig. 3. An example of timing in the path metric update circuitry of the canonic cascade Viterbi decoder with binary alphabet and constraint length 4. The latency of each processor is assumed equal to one clock cycle.

tion. With the exception of the last processor in the ring, $PE_{\nu-1}$, each processor PE_j is followed by two shift registers of length 2^j, and a cross-point switch, SW_j. The last processor, $PE_{\nu-1}$, is followed by a special switch $SW_{\nu-1}$, which can be thought of as a special dual port memory with a controller dedicated to reordering the state information flow from $PE_{\nu-1}$ to PE_0. The efficient implementation of this reordering dual port memory is discussed in Section VIII.

In addition, branch generation circuitry consists of two parts: a received symbol FIFO buffer (FIFO in Fig. 2) and a set of branch metric generators (BMG_j), one for each processor (PE_j). Every trellis_stage_time the FIFO buffer receives an n-tuple of quantized channel output symbol, r_i, the output of the FIFO is routed in a rotating fashion to one of the branch metric generators by a synchronous sampler at the time corresponding to the beginning of a new cycle of state transition evaluations in the corresponding processor. Upon receiving a channel output symbol, the branch metric generator will proceed to compute branch metrics for every state transition present in the state transition diagram (trellis). Since shift register sizes between processors are not identical, branch metric generators will fetch symbols from the FIFO queue at intervals that are not evenly distributed in time, thus the use of a FIFO queue (i.e., an elastic buffer), and not just a shift register is required.

Finally, survivor sequence storage and retrieval circuitry is required to produce the decoded sequence. Two methods are available: the register exchange method and the traceback method. The register exchange method is conceptually simpler, while the traceback method consumes less area and wiring bandwidth. Either one of two survivor sequence management schemes can be used in a canonic cascade Viterbi decoder. Survivor sequence management techniques are further discussed in [17], [19].

A minor modification of the CCVD is possible, where shift register sizes between processors are placed in order of decreasing length; this is analogous to switching between decimation-in-time and decimation-in-frequency algorithms for the fast Fourier transform. The scheme depends only on a definition of the most significant bit in the convolutional encoder's shift register. Throughout this paper, an assumption will be made that the data in the convolutional encoder is shifted from the most significant bit toward the least significant bit, but the alternative definition would work just as well

Fig. 3 provides a detailed illustration of the operation in the ACS data-path corresponding to the CCVD architecture of Fig. 2. All the state path metrics associated with the same stage of the trellis are updated in one processor. The numbers inside boxes that represent processors are not actual path metrics, rather they represent the state number. For instance, numbers 0 and 1 on the left and 0 and 8 on the right indicate that the path metrics of states 0 and 8 of a given trellis_stage_time are computed from the path metrics of the states 0 and 1 of the previous trellis_stage_time. Each successive processor begins computations before the previous processor has completed its computations, so that stages associated with the next stage of the trellis can be processed even before all states associated with the previous stage have been computed. Note that computations associated with different stages of the trellis are staged unevenly: whereas PE_1 starts computing two clock cycles after PE_0, PE_2 has to be delayed by three more clock cycles, and PE_3 by five more. This explains the necessity of using FIFO buffering in the sampled data input section. The switching control algorithm for SW_0 through $SW_{\nu-2}$ is straightforward: each switch, SW_j, alternates between straight-through and criss-cross configurations with a period of 2^j, while switch $SW_{\nu-1}$ is significantly more complex, and can best be described as a dual-port memory with a controller. Writing and reading is done according to a predetermined algorithm to perform re-ordering of the state information. This allows input to PE_0 to arrive in proper order [9]. The special $SW_{\nu-1}$ will be analyzed in greater detail in Section VIII.

When operating in the manner described, the cascade decoder decodes ν bits every 2^ν clock cycles. It is apparent from Fig. 3 that each of the ν processors in the system will be idle for one half of the time, leading to a speedup of $\nu/2$ compared to a uniprocessor Viterbi decoder. It may be possible to increase the utilization of the CCVD by using it to decode two interleaved streams of data. One can also ascertain that each storage location within the CCVD is only utilized one half of the time, and, since the total amount of memory storage required is equal to the number of states, 2^ν, times the number of bits required to store one path metric, w_{pm}, the total size of the path metric memory inside the CCVD is $2 \times 2^\nu \times w_{pm}$.[1]

An extension of the CCVD to arbitrary, nonbinary (q-ary) input symbol alphabets is straightforward and closely related to the radix-q extension of the fast Fourier transform [15]. Utilization of the processors and path metric storage locations in the cases of a q-ary alphabet will be $1/q$.

V. The Folded Cascade Architecture

In the previous section we have pointed out that the CCVD implementation can be efficiently laid out by exploiting local wiring and replication of the processors, switches, and shift registers. However, utilization of the storage elements and the path metric computation circuitry is low ($1/2$ for the binary alphabet CCVD and $1/q$ for the q-ary alphabets). Let us consider the following question: is it possible to modify the CCVD so that each processor is utilized 100% of the time? One potential solution is to employ half as many processors, each one fully utilized.

This is the basis for the folded cascade Viterbi decoder (FCVD). Rigorous proof that causality is nowhere violated, and that in every stage every state will be computed before it is used in the next stage will be supplied in the next section. Fig. 4 demonstrates how computations for the 16-state Viterbi decoder can be performed on two processors instead of four, as illustrated in Fig. 3. The new design utilizes all processors and memory locations 100% of the time, while keeping all the advantages of the CCVD (regular structure, modularity, local communications).

Having discovered the CCVD and FCVD architectures (with ν and $\nu/2$ processors respectively), it is natural to ask whether other similar architectures exist with a number of processors that is not equal to ν or $\nu/2$. In the next section we will demonstrate that such architectures do indeed exist and derive their general properties, including the hardware utilization and speedup available.

VI. Formal Derivation of the Generalized Cascade Architecture for Viterbi Decoders

A. Interstage Delays in Cascade Viterbi Decoders

Before performing the formal derivation of the GCVD, it is helpful to examine the interstage delays in a cascade Viterbi decoder. We have already referred to the not-in-place nature of the computations. We will now elaborate further on this property to derive the general expression for the interstage delays. This expression will be exploited later in this section.

[1]Here we ignore the fact that some memory locations inside the special switch $SW_{\nu-1}$ are empty for more than half of the time. There exists an alternative implementation of the special switch that, while simpler conceptually (dual-port memory with a special sequencer), requires a much more complex illustration. It is sufficient to note that an equivalent design exists, and that in dual-port memory design all memory locations will be used $1/2$ of the time.

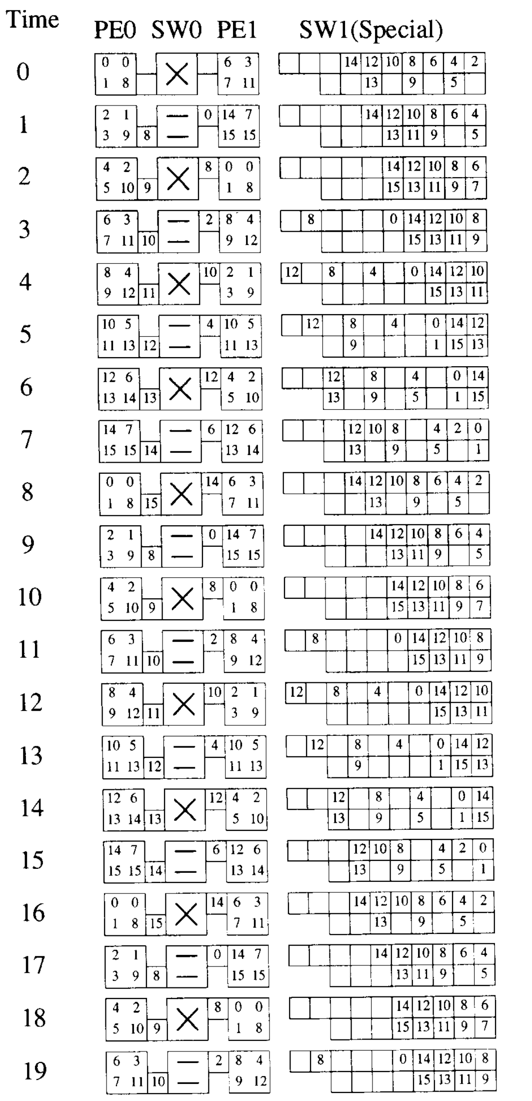


Fig. 4. An example of timing in the path metric update circuitry of the folded cascade Viterbi decoder with binary alphabet and constraint length 4. The latency of each processor is assumed equal to one clock cycle.

Let us adopt the following notation for the binary representation of the state number: $\{a_{\nu-1}a_{\nu-2} \cdots a_2a_1a_0\}$.

Definition 1: Let us define local time (**lt**) as the number of clock cycles from the beginning of the first computation for a given stage of the trellis. Since all stages of the trellis are uniquely mapped to the processors, with each stage assigned to a processor equal to the stage number modulo the number of processors ($\nu/2$ in the case of the FCVD, ν for the CCVD), we can also refer to **lt** as the local time of the processor during the particular stage of computations.

Consider an example of a Viterbi decoder with $q = 2$, $\nu = 4$. The trellis diagram for this decoder, with the butterfly number indicated, is shown in Fig. 5. Suppose that all path metrics begin in natural increasing order of state

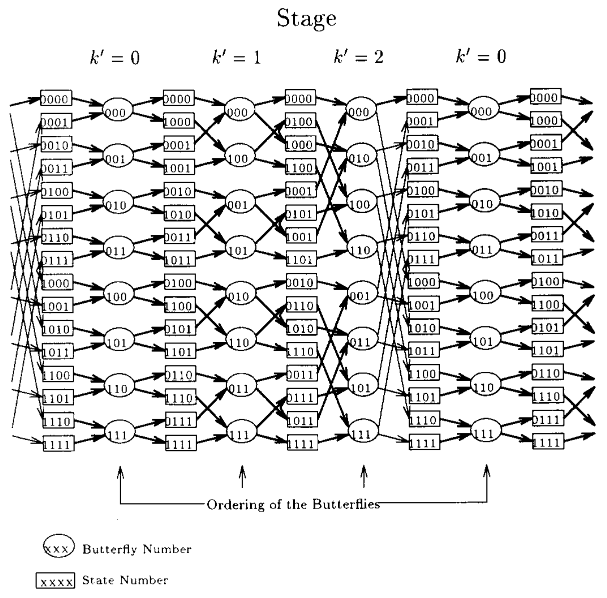


Fig. 5. A trellis diagram of the state transitions for the Viterbi decoder with $q = 2$, $\nu = 4$. State numbers and butterfly numbers are as indicated. $k' = k \bmod (\nu - 1)$, where k is the stage number; the trellis is periodic with a period $\nu - 1$.

numbers, as shown in Fig. 5. The first evaluation utilizes path metrics for states {0000} and {0001} to produce path metrics for states {0000} and {1000}. Note that while we are manipulating the path metrics, their ordering depends on the state number and is independent of the actual value of the path metric. Ordering of the path metrics after zero, one, two, and three stages of the decoding is illustrated in Fig. 5. Suppose we wish to begin performing computations of stage one as soon as possible after the computed path metrics begin to arrive from the zeroth stage. It is apparent that the processor for stage one cannot begin computations, immediately after the first two metrics for states {0000} and {1000} have arrived, since the path metrics for the states {0001} and {1001} have not yet been computed; it is necessary to wait for one more clock cycle until these path metrics become available. This wait time increases to two clock cycles for the second stage and four clock cycles for the third stage (plus the latency of the ACS).

In the zeroth stage the path metrics of the states are used in the natural order. The path metrics leave the zeroth stage in the order derived from their original order by circularly shifting the state number by one bit to the right. This is a direct consequence of the way the convolutional encoder operates. Recall that the contents of the shift register in the convolutional encoder are shifted to the right every clock cycle, with the LSB (a 1 or a 0) shifted out and a new MSB shifted in. Since all stages of the Viterbi decoder are identical, we can derive the ordering of the path metrics generated for any number of stages by performing the appropriate number of circular rotations on the state number. Consider the general case of the Viterbi decoder with a binary alphabet ($q = 2$) and $N = 2^{\nu}$ states. We can give the following definition of the schedule for the cascade Viterbi decoder:

Definition 2: A schedule for any Viterbi decoder is defined by the time when the inputs (state path metrics) for a particular butterfly computation are required.

For a generalized cascade Viterbi decoder we choose a particular schedule such that the local time at which the inputs of a particular butterfly are required is a circular rotation to the left of the bits representing the butterfly number. The number of bit positions by which the bit representation of a butterfly number must be rotated is equal to $j' \equiv j \bmod (\nu - 1)$.

In particular, computation of the ACS butterfly $\{a_{\nu-1}a_{\nu-2} \cdots a_2a_1\}$ in the jth stage will begin at local time

$$\mathbf{lt} = \{a_{\nu-j'-1}a_{\nu-j'-2} \cdots a_2a_1a_{\nu-1} \cdots a_{\nu-j'}\}.$$

Definition 3: For a pipelined processor, butterfly computations start every clock cycle. If the pipeline is d stages deep (latency $= d$) then the output from the ACS butterfly $\{a_{\nu-1}a_{\nu-2} \cdots a_2a_1\}$ in the jth stage will be produced at

$$\mathbf{lt} = \{a_{\nu-j'-1}a_{\nu-j'-2} \cdots a_2a_1a_{\nu-1} \cdots a_{\nu-j'}\} + d.$$

Or, equivalently, using the one-to-one correlation between the butterfly number and the state numbers of the inputs and the outputs of that butterfly, we can state that in any stage j of the cascade Viterbi decoder the time at which path metrics of the states $\{a_{\nu-1}a_{\nu-2} \cdots a_2a_1 0\}$ and $\{a_{\nu-1}a_{\nu-2} \cdots a_2a_1 1\}$ are required is at local time

$$\mathbf{lt} = \{a_{\nu-j'-1}a_{\nu-j'-2} \cdots a_2a_1a_{\nu-1} \cdots a_{\nu-j'}\}$$

and the local time at which the path metrics of the states $\{0a_{\nu-1}a_{\nu-2} \cdots a_2a_1\}$ and $\{1a_{\nu-1}a_{\nu-2} \cdots a_2a_1\}$ are available from stage j is

$$\mathbf{lt} = \{a_{\nu-j'-1}a_{\nu-j'-2} \cdots a_2a_1a_{\nu-1} \cdots a_{\nu-j'}\} + d$$

where d is the latency (number of pipeline stages) of the processor, $j' \equiv j \bmod (\nu - 1)$.

The schedule that defines the cascade Viterbi decoder is just one of many equivalent (in terms of performance and resultant implementation) possible schedules.

We have briefly mentioned the fact that computations in stage $j + 1$ cannot start immediately after the first pair of state metrics have become available from stage j. Definitions 1 through 3 allow us to derive the minimum necessary delay between stages j and $j + 1$ as a function of the stage number j.

Definition 4: The interstage delay (δ_j) is the minimum number of clock cycles between the time the first pair of path metrics is available from the jth stage and the time the first computation in the $j + 1$st stage is able to begin.

Theorem 1: For the GCVD with k processors, the interstage delay, δ_j, $0 \le j \le k - 1$, is equal to $2^{j \bmod (\nu - 1)}$ and guarantees that the computations in the $j + 1$st stage can proceed in a pipelined fashion, with a new set of inputs available every clock cycle until all computations of the $j + 1$st stage are completed.

For proof of this theorem refer to [17].

From our discussion of the causality condition in Section III and Theorem 1 it follows that the GCVD schedule is the optimum (or perhaps one of the many equivalent optimum schedules), since computations in every stage begin as soon as permitted by the causality condition and computation of every stage of the trellis proceed without interruptions until all ACS operations of a given stage of the trellis have been performed.

B. Timing of the Generalized Cascade Architecture

In this section we proceed to investigate the ordering of operations in the complete ACS datapath, and consequently move from consideration of local times (**lt**) to a global perspective. To differentiate from local time of the previous subsection, we use the designation global time (**gt**) to refer to the overall time index of the system. For instance the global time (**gt**) is entirely equivalent to the time index in Figs. 3 and 4. We can now proceed to evaluate the global time (**gt**) at which a given state path metric will be available from stage j, and to show the delay required between $\text{PE}_{\nu-1}$ and PE_0 so that the path metrics can be recirculated. The global time will consist of three parts: the combined interstage delay of all the stages from 0 to $j - 1$; the combined latencies of the stages 0 through $j - 1$, and the local time at which the path metric of a given state is evaluated in the jth stage.

The sum of the interstage delays is $\Sigma_{i=0}^{j-1} 2^{i'}$.

The sum of the latencies for stages 0 through $j - 1$ is

$$\sum_{i=0}^{j-1} d = j \times d.$$

Finally, the local time of the jth stage at which a given state path metric will become available is

$$\mathbf{lt} = \{a_{\nu-j'-1}a_{\nu-j'-2} \cdots a_2a_1a_{\nu-1} \cdots a_{\nu-j'}\} + d.$$

Thus the global time at which path metrics of the states $\{0a_{\nu-1}a_{\nu-2} \cdots a_2a_1\}$ and $\{1a_{\nu-1}a_{\nu-2} \cdots a_2a_1\}$ will be available from stage $j - 1$ is

$$\mathbf{gt}_1 = \sum_{i=0}^{j-1} 2^{i'} + (j + 1)d + \{a_{\nu-j'-1}a_{\nu-j'-2} \cdots a_2a_1a_{\nu-1} \cdots a_{\nu-j'}\}. \quad (1)$$

These path metrics will now be forwarded to PE_0 for the purpose of computing the path metrics of stage j. PE_0 is busy with the path metrics of stage 0 up to $\mathbf{gt} = 2^{\nu-1}$. It can then begin to accept the inputs for stage ν in the same order as that used for stage 0 (i.e., in the natural increasing order of state numbers). Let $\mathbf{gt}_2$ be the time at which PE_0 can accept path metrics of state $\{\psi a_{\nu-1}a_{\nu-2} \cdots a_2a_1\}$

$$\mathbf{gt}_2 = 2^{\nu-1} + \{\psi a_{\nu-1}a_{\nu-2} \cdots a_2\}. \quad (2)$$

To maintain causality we require that the information be produced before it is consumed, therefore $\mathbf{gt}_2 \geq \mathbf{gt}_1$ for any state. The worst case state can be found by using the following rule: If a bit a_k occupies a higher bit position in the $\mathbf{gt}_1$ expression than in the $\mathbf{gt}_2$ expression, then set it to a 1, otherwise set it to a 0. Finally set ψ to a 0, since this choice of ψ makes meeting the criterion $\mathbf{gt}_2 \geq \mathbf{gt}_1$ more difficult (i.e., minimize the expression on the left hand side of the inequality, if the inequality is true when $\psi = 0$ then it is also guaranteed to be true when $\psi = 1$, but the opposite is not always true).

We will compute the criteria for inequality $\mathbf{gt}_2 \geq \mathbf{gt}_1$ to be true by first considering the case of $1 \leq k < \nu$, followed by the more general case of k any integer greater than or equal to one.

Let us substitute $j = k - 1$, $(1 \leq k < \nu)$ into (1), which gives the following value of the global time at which the path metrics of the state $\{\psi a_{\nu-1} a_{\nu-2} \cdots a_2 a_1\}$ (where ψ can be a 0 or a 1) will become available from processor $k - 1$:

$$\mathbf{gt}_1 = 2^{k-1} + k \times d - 1 + \{a_{\nu-k} a_{\nu-k-1} \cdots a_2 a_1 a_{\nu-1} a_{\nu-2} \cdots a_{\nu-k+1}\}. \tag{3}$$

The global time at which the path metric of the state $\{\psi a_{\nu-1} a_{\nu-2} \cdots a_2 a_1\}$ will be required at the inputs of processor 0 will be the same as in (2) or

$$\mathbf{gt}_2 = 2^{\nu-1} + \{\psi a_{\nu-1} a_{\nu-2} \cdots a_2\}. \tag{4}$$

The worst case state will again be where $\psi = 0$, set a bit a_i to a 1 if it occupies a higher bit position in the $\mathbf{gt}_1$ expression than in the $\mathbf{gt}_2$ expression, otherwise set bit a_i to a 0. From comparison of (3) with (4) it is apparent that bits $a_{\nu-k}, a_{\nu-k-1}, \cdots, a_2, a_1$ must be set to a 1 and the rest of the bits to zero.

Substituting these values into inequality $\mathbf{gt}_2 > \mathbf{gt}_1$ gives

$$2^{\nu-1} + \Big\{\overbrace{00 \cdots 00}^{k}\ \overbrace{11 \cdots 11}^{\nu-k-1}\Big\} \geq 2^{k-1} + k \times d + \Big\{\overbrace{11 \cdots 11}^{\nu-k}\ \overbrace{00 \cdots 00}^{k-1}\Big\} - 1. \tag{5}$$

The left-hand side of this inequality will become

$$2^{\nu-1} + \Big\{\overbrace{00 \cdots 00}^{k}\ \overbrace{11 \cdots 11}^{\nu-k-1}\Big\} = 2^{\nu-1} + 2^{\nu-k-1} - 1. \tag{6}$$

The right-hand side of this inequality will become

$$2^{k-1} + k \times d + \Big\{\overbrace{11 \cdots 11}^{\nu-k}\ \overbrace{00 \cdots 00}^{k-1}\Big\} - 1 = 2^{\nu-1} + k \times d - 1. \tag{7}$$

We can now restate Inequality 5 using the results of (6) and (7) and solve for d

$$d \leq \frac{2^{\nu-k-1}}{k}. \tag{8}$$

We may conclude now that for a constraint length ν cascade Viterbi decoder an implementation with k processors $(1 \leq k \leq \nu - 1)$ exists providing $\lfloor 2^{\nu-k-1}/k \rfloor \geq 1$, and this implementation utilizes every one of the processors 100% of the time. In addition, the ACS circuitry inside every processor can be pipelined with up to $\lfloor 2^{\nu-k-1}/k \rfloor$ pipeline stages available in the data path. This is extremely important, since pipelining allows us to increase the clock rate (and throughput) of the Viterbi decoder. This increase in throughput due to pipelining is multiplied by the throughput increase that is made possible by utilizing k processors running in parallel, with each processor utilized 100% of the time.

If the number of processors (k) selected leads to Inequality 8 bring violated, then an extra delay, Δ, must be added between the beginning of the last computation associated with stage i in $\mathrm{PE}_{i \bmod k}$ and the beginning of the first computation associated with stage $i + k$ in $\mathrm{PE}_{i \bmod k}$. The amount of extra delay is $\Delta = k \times d - 2^{\nu-k-1}$.

Definition 5: The utilization $U(k, d)$, when decoding a single data stream, is equal to the number of clock cycles required to update all the path metrics associated with a given trellis_stage_time, T_u, divided by $T_u + \Delta$.

The operation of a GCVD will consist of alternating periods of performing path metric updates for $T_u = 2^{\nu-1}$ clock cycles and an idle period (perhaps of length zero) lasting $\Delta = k \times d - 2^{\nu-k-1}$. The utilization of each processor will be

$$U(k, d) = \min \begin{cases} 1 \\ \dfrac{2^{\nu-1}}{2^{\nu-1} + k \times d - 2^{\nu-k-1}}. \end{cases} \tag{9}$$

It is important to note here that this expression for the utilization of the GCVD is only true for $1 \leq k \leq \nu - 1$. Thus it is necessary to analyze the utilization of the GCVD with $k \geq \nu$ separately.

We can also derive the value of $U(k, d)$ for any $k \geq \nu$. Define $k' \equiv k \bmod (\nu - 1)$. The general expression for $\mathbf{gt}_1$ becomes

$$\mathbf{gt}_1 = \sum_{i=0}^{k-2} 2^{i'} + k \times d + \{a_{\nu-k'} a_{\nu-k'-1} \cdots a_2 a_1 a_{\nu-1} a_{\nu-2} \cdots a_{\nu-k'+1}\}. \tag{10}$$

Equation (4) for $\mathbf{gt}_2$ remains unchanged.

Once again, causality requires that $\mathbf{gt}_2 \geq \mathbf{gt}_1$. The worst case state will again be one where $\psi = 0$, bit a_i set to a 1 if it occupies a higher bit position in the $\mathbf{gt}_1$ expression than in the $\mathbf{gt}_2$ expression, otherwise bit a_i is set to a 0. From a comparison of (10) and (4) it is apparent that bits $a_{\nu-k'}, a_{\nu-k'-1} \cdots, a_2, a_1$ must be set to a 1 and the rest of the bits to zero.

Substituting these values into the inequality $\mathbf{gt}_2 > \mathbf{gt}_1$

gives

$$2^{\nu-1} + \left\{ \overbrace{00\cdots00}^{k'} \overbrace{11\cdots11}^{\nu-k'-1} \right\} \geq \sum_{i=0}^{k-2} 2^{i'} + k \times d + \left\{ \overbrace{11\cdots11}^{\nu-k'} \overbrace{00\cdots00}^{k'-1} \right\}. \qquad (11)$$

The left-hand side of this inequality will become

$$2^{\nu-1} + \left\{ \overbrace{00\cdots00}^{k'} \overbrace{11\cdots11}^{\nu-k'-1} \right\} = 2^{\nu-1} + 2^{\nu-k'-1} - 1. \qquad (12)$$

The right-hand side of this inequality will become

$$\sum_{i=0}^{k-2} 2^{i'} + k \times d + \left\{ \overbrace{11\cdots11}^{\nu-k'} \overbrace{00\cdots00}^{k'-1} \right\} = 2^{\nu-1} + k \times d - 1 + \left\lfloor \frac{k}{\nu-1} \right\rfloor \times (2^{\nu-1} - 1). \qquad (13)$$

We can now restate Inequality 11 using the results of (12) and (13) and solve for d

$$2^{\nu-1} + 2^{\nu-k'-1} - 1 \geq 2^{\nu-1} + k \times d - 1 + \left\lfloor \frac{k}{\nu-1} \right\rfloor \times (2^{\nu-1} - 1)$$

$$2^{\nu-k'-1} \geq k \times d + \left\lfloor \frac{k}{\nu-1} \right\rfloor \times (2^{\nu-1} - 1)$$

$$d \leq \frac{2^{\nu-k'-1} - \left\lfloor \dfrac{k}{\nu-1} \right\rfloor \times (2^{\nu-1} - 1)}{k}. \qquad (14)$$

It is apparent that no positive value of d will satisfy Inequality 14 when $k \geq \nu$, therefore extra delay

$$\Delta \geq \left\lfloor \frac{k}{\nu-1} \right\rfloor \times (2^{\nu-1} - 1) - 2^{\nu-k'-1} + k \times d$$

must be added between the completion of the zeroth stage evaluation in the zeroth processor and the beginning of the evaluation of stage k in the zeroth processor. Once again, it is possible to increase the delay, simultaneously increase the number of pipeline stages available, and process multiple streams of data in parallel, thus bringing utilization back to 100%. It is possible to determine how big Δ must be made to decode any number of independent streams of data in an interleaved fashion. To decode n interleaved streams it is necessary to set $\Delta \geq (n - 1) \times T_u$, with $\Delta = (n - 1) \times T_u$ being a condition for being able to decode n data streams with 100% utilization. Of course some values of n may be impossible to achieve, since the condition

$$\Delta \geq \left\lfloor \frac{k}{\nu-1} \right\rfloor \times (2^{\nu-1} - 1) - 2^{\nu-k'-1} + k \times d$$

must be satisfied at all times to ensure causality is not violated.

Note that the first term in the expression for Δ dominates, thus increasing the value of k by 1 changes the value of Δ by a small amount only, unless increasing k by 1 will cause an increase in $\lfloor k/(\nu - 1) \rfloor$, which increases the value of Δ by a large amount. The most obvious example of this occurs when k is increased from $\nu - 1$ to ν. Consider Fig. 3, where a large delay is required if path metrics are forwarded to PE_0 from PE_3; yet if we were to remove PE_3, and the path metrics were forwarded to PE_0 from PE_2 the net result would be a much smaller number of clock cycles during which a given processor is idle (shorter delay), resulting in a better processor utilization.

We can now use the expression for Δ to derive a general equation for utilization possible when decoding a single stream of data for all values of k:

$$U(k, d) = \min \left\{ 1, \; \frac{2^{\nu-1}}{2^{\nu-1} + k \times d - 2^{\nu-k'-1} + \left\lfloor \dfrac{k}{\nu-1} \right\rfloor \times (2^{\nu-1} - 1)} \right\}. \qquad (15)$$

Equation (15) is true for all values of $k > 0$, and for $k \leq \nu - 1$ it matches (9), as expected.

An extension of the results of this section for a Viterbi decoder with a nonbinary input alphabet is presented in [17].

VII. Speedup and Processor Utilization in GCVD

In Fig. 6 the maximum speedup (product of the number of processors k and the utilization $U(k, d)$ available is plotted against the number of processors k for two particular cases $\nu = 14$ with pipelining depth $d = 17$ and $\nu = 7$ with pipelining depth $d = 4$. It is apparent that when $\nu = 14$ even for the large value of $d = 17$, the speedup is nearly linear with the number of processors k up to and including $k = \nu - 1$ ($k = 13$ in this particular case) where the speedup has its maximum value of 12.7. Note that in addition to the primary linear speedup region which runs from $k = 1$ to $k = \nu - 1$, there are also secondary, tertiary, and so forth linear speedup regions. As is apparent from (15), these linear regions will have a slope that is inversely proportional to $1 + \lfloor k/(\nu - 1) \rfloor$, indicating poor utilization in decoding one stream of data. On the other hand, n interleaved streams of data ($n \geq 1 + \lfloor k/(\nu$

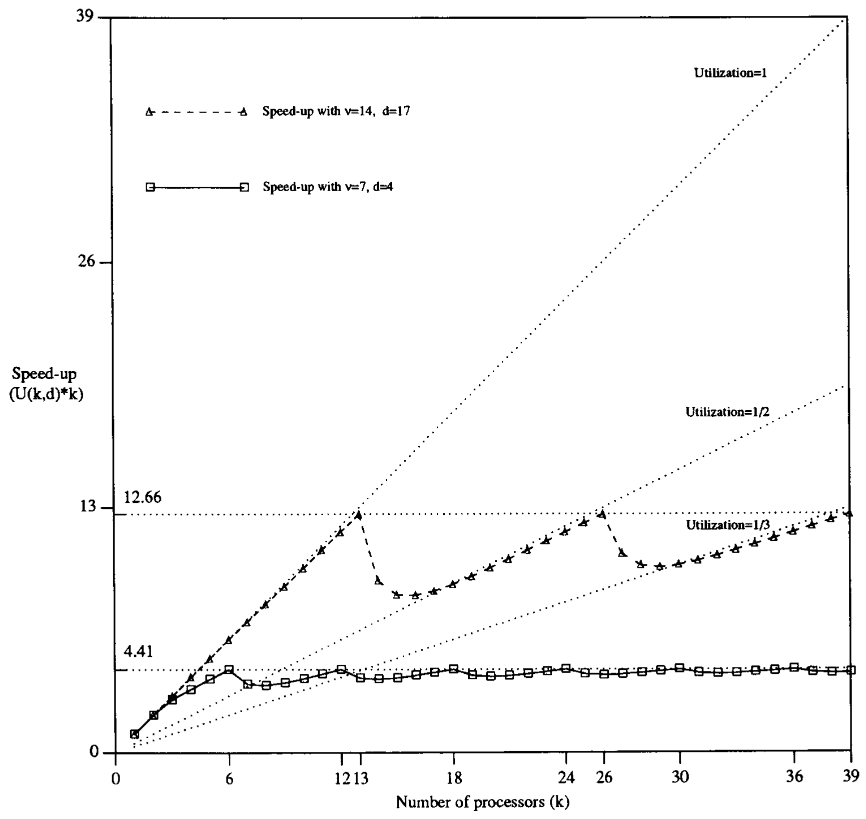


Fig. 6. Plot of speedup that can be obtained using GCVD with number of processors varying from 1 to 39 for constraint lengths $\nu = 7$ (pipelining depth $d = 4$) and $\nu = 14$ ($d = 17$).

$- 1)\rfloor$) can be decoded with utilization that is very close to 100%.

As the constraint length ν increases, the speedup curve will preserve its general shape; deviation of the speedup curve from the straight lines indicating primary, secondary, and so forth linear speedup regions will actually decrease.

VIII. Interstage Shift Registers and Cross-Point Switches

Recall from our discussion in Section VI that a k-processor GCVD contains $k - 1$ cross-point switches and that two shift registers are associated with every cross-point switch; the length of each shift register following the jth processor is 2^j, $0 \leq j \leq k - 2$. The design of these modules is straightforward. One circuit deserves special attention: a special switch, SW_{k-1} that is used to reorder the path metrics before recirculating them back to the inputs of the zeroth stage. We have stated in Section IV that conceptually we can consider this reordering to be performed in a dual-port memory with a special sequencer that ensures proper retrieval of path metrics.

Designing a dual-port memory with an appropriate sequencer is possible, but not trivial, yet, we already use a number of circuits that perform reordering of path metrics based on our knowledge of scheduling in a given stage of a GCVD. These circuits are the other cross-point switches with their attendant shift registers. Admittedly their job is somewhat simpler, since they do not perform reordering like that required at the inputs to the zeroth stage. Yet it seems plausible that one or more cross-point switches may be able to accomplish the reordering required of the dual-port memory. In this section we will demonstrate that a combination of one or more cross-point switches with shift registers can always implement the necessary reordering of the path metrics. For any combination of the constraint length ν and the number of processors k we will compute the number of cross-point switches required and the size of the shift registers associated with each cross-point switch. Finally, we will demonstrate that, through elimination of the equal sections of the shift registers on the upper and lower paths between the switches, it is possible to eliminate the delay on the critical path. Alternatively, we can keep at least one shift register between any two switches; the reduction in the processor utilization in this case is equal to the reduction that would be caused by increasing the pipelining depth of each processor by a few (typically one) stages. A GCVD with $k = \nu - 1$ processors requires only a single cross-point switch to implement the reordering of the path metrics.

Full analysis of the construction of the recirculation network is quite lengthy and can be found in [17]. We will instead give a summary of the method of construction.

Three distinct cases may be encountered in designing a concatenation of switches:

1) Number of processors $k = \nu - 1$. A single cross-point switch with registers of length $2^{\nu-2}$ is required.

2) Number of processors, k, and $\nu - 1$ are relatively prime. In this case, exactly $\nu - 1$ switches are required. The length of the shift registers associated with each switch can be computed from the following formula: each register associated with ith switch is of length $2^{(\nu-2-i \times k)'}$.

3) Number of processors, k, and $\nu - 1$ are not relatively prime, with $\gcd(k, \nu - 1) = m$. In this case $\nu - 1 + m - 1$ switches are required. The method for computing lengths of shift registers associated with each switch is discussed in [17].

Consider three concatenated switches with shift registers illustrated in Fig. 7. It appears that this combination of switches will delay the path metric of any state. Yet in our analysis of GCVD timing in Section VI we have assumed that the path metric on a critical path—the one with the worst (possibly negative) difference between the local time of its generation in the jth stage and the local time of its consumption in the zeroth stage (after recirculation) is sent to the inputs of zeroth stage with no delay. Fortunately there is a simple modification to our multistage switch network that allows for a zero delay passing of the state path metric on a critical path. Consider a pair of shift registers located between the first and second cross-point switches in Fig. 7. In general, the lengths of the two registers will not be equal. Suppose, without loss of generality, that the length of the upper shift register is 2^l, the length of the lower shift register is 2^m, and $m < l$. We may shorten each shift register by 2^m, leaving a shift register of length $(2^l - 2^m)$ in the upper path and a shift register of length $(2^m - 2^m) = 0$ (or simply a wire) in the lower path. Since removal of equal delays does not affect relative timing of the state path metrics taking the upper and the lower paths, the overall effect of reordering remains in place. At the same time, a delay-free path becomes available for the state path metric on a critical path, as shown in Fig. 7. In a real circuit, the nonideal switches may cause a problem when the path metric on a critical path is routed by multiple cross-point switches without being latched in between. This problem can be avoided by removing fewer delay registers between any two switches. This will have exactly the same effect on the overall timing as would an increase in a pipelining depth of the processors. One delay register between every pair of switches on the route taken by the path metric on a critical path will be sufficient to restore signal levels following each switch, yet because the total delay will be equal to the number of switches $O(\nu)$, the result will be equivalent to that of increasing the pipelining depth of each processor by at most ν (in the case of the uniprocessor GCVD). No increase in the apparent pipelining depth of a GCVD with $\nu - 1$ stages will occur, because only a single switch is required to perform the reordering of the state path metrics in a GCVD with $\nu - 1$ stages. Simplicity of recirculation network design is an added bonus to the previously discussed advantages of a $(\nu - 1)$-stage GCVD.

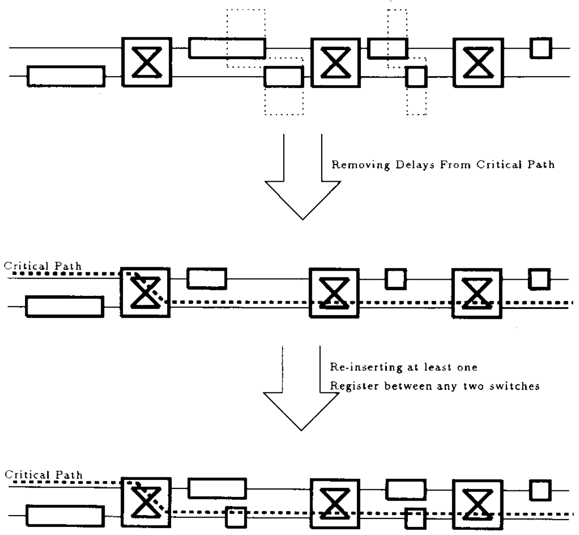

Removing Delays From Critical Path
Critical Path
Re-inserting at least one
Register between any two switches
Critical Path

Fig. 7. Removal of common delays from the switching network achieves correct absolute as well as relative timing of the state path metrics.

IX. Discussion

In our discussion of GCVD architecture we have ignored the survivor sequence memory management design. The best possible design of the sequence memory management for the GCVD is a one-pointer [18], [19] with distributed memory: total decision memory is split into $\nu - 1$ memory banks (one for each ACS unit); each memory bank is associated with an ACS unit. The distributed memory structure allows decision writing to proceed over the local wires, while slower traceback/decode read operation travels around the ring of ACS units (but in the direction opposite that of the path metric). An added benefit of the distributed memory is the fact that smaller memory modules can be operated at higher speed.

Finally, it is instructive to compare the GCVD architecture with other architectures that have been used for Viterbi decoding. One of the authors has shown [17] that GCVD architecture is capable of achieving a throughput increase (vis-à-vis a uniprocessor) that grows linearly with both the number of processors and the VLSI area required. This compares favorably with other architectures [9] (square mesh, shuffle-exchange) for which throughput grows as a square root of the VLSI area required.

X. Summary

In this paper we have demonstrated the existence of a family of generalized cascade Viterbi decoder architectures that can be implemented as a ring structure with unidirectional local communications for a binary alphabet. An extension to a case of any q-ary input alphabet is also possible [17]. The proposed family of architectures is well suited for VLSI implementation.

We have demonstrated that the GCVD architecture is capable of efficiently utilizing from one to $\nu - 1$ processors in decoding a single stream of data. We have shown that for the generalized cascade Viterbi decoders will a small number of stages, it is possible to achieve full utilization of all processors and simultaneously pipeline the path metric update circuitry of the processors. This makes the GCVD architecture attractive for implementing the Viterbi algorithm where a large constraint length, ν, and a high throughput rate are required. For larger values of the number of processors it is impossible to achieve full utilization in decoding a single stream of data, but full utilization may still be achieved in decoding multiple interleaved streams of data. A GCVD with $k = \nu - 1$ processors is especially attractive because it achieves the highest speedup possible in decoding a single stream of data, with nearly 100% utilization. Furthermore, a recirculation network for a GCVD with $k = \nu - 1$ stages is guaranteed to be simpler than a recirculation network for a GCVD with any number of stages that is not an integer multiple of $\nu - 1$. Similarly, all GCVDs with k equal to an integer multiple of $\nu - 1$ will provide the largest speedup in decoding multiple interleaved streams of data.

Acknowledgment

The authors are grateful to the anonymous referees for their helpful comments.

References

[1] A. J. Viterbi, "Error bounds for convolutional codes and an asymptotically optimum decoding algorithm," *IEEE Trans. Inform. Theory*, vol. IT-13, vol. 4, pp. 260-269, Apr. 1967.

[2] G. D. Forney, Jr., "The Viterbi algorithm," *Proc. IEEE*, vol. 61, pp. 268-278, Mar. 1973.

[3] S. Dolinar, "A new code for Galileo," TDA Progress Rep. 42-93, Jet Propulsion Lab., Pasadena, CA, Jan.-Mar. 1988.

[4] J. Statman, G. Zimmerman, F. Pollara, and O. Collins, "A long constraint length VLSI Viterbi decoder for the DSN," TDA Progress Rep. 42-95, Jet Propulsion Lab., Pasadena, CA, July-Sept. 1988.

[5] K. S. Gilhousen *et al.*, "Coding systems study for high data rate telemetry links," Tech. Rep., NASA, Jan. 1971 (prepared by Linkabit Corp. under Contract NAS2-6024).

[6] C. B. Shung *et al.*, "Implementation issues for the design of a rate 8/10 trellis code for partial response channels," presented at the Third IBM Workshop ECC, San Jose, CA, Sept. 1989.

[7] G. D. Forney, Jr., "Maximum likelihood sequence estimation of digital sequences in the presence of intersymbol interference," *IEEE Trans. Inform. Theory*, vol. IT-18, pp. 363-378, May 1972.

[8] C. Y. Chang and K. Yao, "Viterbi decoding by systolic array," in *Proc. Twenty-Third Annu. Allerton Conf. Commun., Contr., Computing*, Monticello, IL, Oct. 1985, pp. 430-439.

[9] P. G. Gulak and T. Kailath, "Locally connected VLSI architectures for the Viterbi algorithm," *IEEE J. Select. Areas Commun.*, vol. 6, pp. 527-538, Apr. 1988.

[10] P. G. Gulak and E. Shwedyk, "VLSI structures for Viterbi receivers: Part I-General theory and applications," *IEEE J. Select. Areas Commun.*, vol. 4, pp. 142-154, Jan. 1986.

[11] O. Collins, F. Pollara, S. Dolinar, and J. Statman, "Wiring Viterbi decoders (splitting de Bruijn graphs)," TDA Progress Rep. 42-96, Jet Propulsion Lab., Pasadena, CA, Oct.-Dec. 1988.

[12] H. D. Lin and D. G. Messerschmitt, "Improving the iteration bound of finite state machines," in *Proc. ISCAS*, May 1989, pp. 1328-1331.

[13] K. K. Parhi, "Look-ahead in dynamic programming and quantizer loops," In *Proc. ISCAS*, May 1989, pp. 1382-1387.

[14] G. Fettweis and H. Meyr, "Parallel Viterbi algorithm implementation: Breaking the ACS-Bottleneck," *IEEE Trans. Commun.*, vol. 37, pp. 785-789, Aug. 1989.

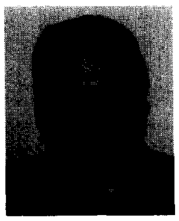
[15] L. Rabiner and B. Gold, *Theory and Applications of Digital Signal Processing*. Toronto, Canada: Prentice-Hall of Canada, 1975.

[16] C. D. Thompson, "A complexity theory for VLSI," Ph.D. dissertation, Carnegie-Mellon Univ., Pittsburgh, PA, 1980.

[17] G. Feygin, "A multiprocessor architecture for Viterbi decoders with linear speedup," master's thesis, Univ. Toronto, Toronto, Canada, 1990.

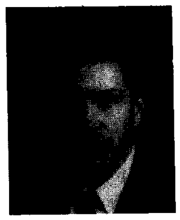
[18] G. Feygin, P. G. Gulak, and F. Pollara, "Survivor sequence memory management in Viterbi decoders," presented at the Third IBM Workshop ECC, San Jose, CA, Sept. 1989.

[19] G. Feygin and P. G. Gulak, "Architectural tradeoffs for survivor sequence memory management in Viterbi decoders," *IEEE Trans. Commun.*, vol. 41, no. 3, pp. 425-429, Mar. 1993.



Gennady Feygin received the B.Sc.E.E. degree from the University of Alberta, Edmonton, Alberta, Canada, in 1986, and the M.Sc. degree in electrical engineering from the University of Toronto, Toronto, Canada in 1990. He is currently pursuing the Ph.D. degree in the Department of Electrical and Computer Engineering, University of Toronto.

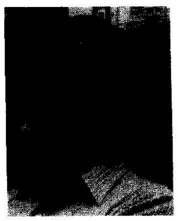
From July 1986 to January 1988 he was a Design Engineer with Amdahl Communications Ltd., Mississauga, Ontario, Canada. His research interests include VLSI architectures for signal processing and digital communications.



Patrick G. Gulak (S'82-M'83) received the Ph.D. degree from the University of Manitoba.

From 1985 to 1988 he was a Research Associate in the Information Systems Laboratory at Stanford University. He is now an Associate Professor in the Department of Electrical and Computer Engineering at the University of Toronto. His research interests are in the areas of circuits, algorithms, and VLSI architectures for digital communications and signal processing applications.

Dr. Gulak has served on the ISSCC program committee (signal processing subcommittee) since 1990. He has twice received the Computer Science Students Union Undergraduate Teaching Award for courses he has taught on computer architecture at the University of Toronto.



Paul Chow (S'79-M'83) received the B.A.Sc. degree with honors in engineering science, and the M.A.Sc. and Ph.D. degrees in electrical engineering from the University of Toronto, Toronto, Ontario, Canada, 1977, 1979, and 1984, respectively.

In 1984 he joined the Computer Systems Laboratory at Stanford University, Stanford, CA, as a Postdoctoral Fellow and later as a Research Associate. He was a major contributor to the MIPS-X project. In January 1988, he joined the Department of Electrical and Computer Engineering at the University of Toronto and is now an Associate Professor. His current research interests include high performance computer architectures, VLSI systems design, and field-programmable gate array architectures and applications.