GENERALIZED CASCADE VITERBI DECODER — A LOCALLY CONNECTED MULTIPROCESSOR WITH LINEAR SPEED-UP

Gennady Feygin Patrick Glenn Gulak Paul Chow

Dept. of Electrical Engineering, University of Toronto, Canada

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 which is directly proportional to the number of processors when the number of processors is smaller than the constraint length of the code ν. The hardware utilization of nearly 100% and availability of deep pipelining inside each processor are demonstrated. An implementation proposal for a constraint length $\nu = 14$ Viterbi Decoder based on the proposed family of architectures is presented. The proposed thirteen-processor decoder is intended to be compatible with specifications for the "Galileo" space probe currently being developed by the Jet Propulsion Laboratory.

1 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.

A number of multiprocessor architectures have been proposed for Viterbi Decoders, including one- and two-dimensional systolic meshes [1] and the perfect shuffle layout [2]. All of these architectures provide throughput increase that is sublinear (i.e. to increase the throughput by a constant factor more than a constant growth in area is required [1]). An architecture that is closely related to the perfect shuffle, known as the crenellated-FFT [3], is currently being implemented at the Jet Propulsion Laboratory. Yet another family of architectures for Viterbi Decoding has been introduced recently by [4], and these architectures are capable of providing the throughput increase that is linear in the number of processors. This family of architectures is based on collapsing multiple stages of a trellis into a single stage of a "super-trellis". Unfortunately, while throughput increase is linear in the number of processing elements, it is also inversely proportional to the number of states in the trellis [4]. Since the number of states in the trellis grow exponentially with the constraint length of the code, this architecture is of benefit only in the case of the short-constraint length codes.

In 1973 Forney [5] pointed out that the trellis diagram of the Viterbi Algorithm differs from the computational flow diagram of the Fast Fourier Transform (FFT) in length only. Recently [1] have been able to take advantage of this relationship between the Viterbi Algorithm and FFT by proposing a novel "cascade" Viterbi Decoder that is closely related to the "cascade" design for the pipelined FFT computation [6], but requires an additional "recirculation" connection from the outputs 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 are also certain shortcomings inherent in this architecture. The utilization of the processors and the state path memory storage elements is only 50%. In addition, a complicated dual-port switch is required to perform the re-ordering of the state path metrics.

2 THE TRELLIS DIAGRAM: MAPPING AND SCHEDULING OF OPERATIONS

Consider a portion of the 16-state ($N = 16$) trellis diagram for the $(2, 1, 4)$ convolutional code illustrated in Figure 1. The nodes in the trellis diagram correspond to the Add-Compare-Select computations and the edges indicate dependencies between the ACS computations of the sucessive 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 which does not violate causality. If more than one such schedule can be found, we may then choose one schedule that maximizes the utilization of the hardware. The solid lines in Figure 1 give an indication of the scheduling constraints imposed by the causality requirement. For instance, in order 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 Figure 1 indicates two ACS computations being performed simultaneously in any stage of the trellis — in a manner reminiscent of the FFT butterfly computation . Since exactly 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, it is advantageous to perform ACS operations two at a time.

[0]The authors gratefully acknowledge financial support provided by the Natural Sciences and Engineering Research Council of Canada and by the Information Technology Research Center of Ontario.

CH2977-7/91/0000-1097 $1.00 © 1991 IEEE

3 THE CANONIC CASCADE ARCHITECTURE

The Canonic Cascade architecture of the Viterbi Decoder was originally introduced in [1]. In the CCVD architecture $log(N)$ processors are arranged in a ring, together with local memory and switches. 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. The Canonic Cascade layout is regular and compact, has only local (nearest neighbour) 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. In the binary input alphabet case each processor ($PE_j, 0 < j < \nu - 1$) consists of two Add-Compare-Select (ACS) circuits in a butterfly configuration. 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 re-ordering the state information flow from $PE_{\nu-1}$ to PE_0. The efficient implementation of this re-ordering dual port memory is discussed in Section 5.

In addition, branch generation circuitry consists of two parts: a Received Symbol FIFO and a set of Branch Generators (BMG_j), one for each Processor (PE_j). Every Trellis_Stage_Time the FIFO queue receives a quantized channel output symbol $\mathbf{r}_i$ (an n-tuple). 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 the new cycle of the 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 [7].

Figure 2(a) provides a detailed illustration of the operation in the ACS datapath corresponding to the CCVD architecture of a CCVD decoder for a binary input alphabet ($q = 2$), constraint length $\nu = 4$, and number of states $N = q^\nu = 16$. 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 staggered 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 shift register with insert capabilities. Shifting and insertion are done according to a predetermined algorithm to perform reordering of the state information. This allows inputs to PE_0 to arrive in proper order [1]. The special switch $SW_{\nu-1}$ will be analyzed in greater detail in Section 5.

When operating in the manner described, the Cascade Decoder decodes ν bits every 2^ν clock cycles. It is apparent from Figure 2(a) that each of the ν Processors in the system will be idle for one-half of the time, leading to a speed-up 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 $2 \times 2^\nu \times w_{pm}$ bits.

An extension of the CCVD to arbitrary, non-binary (q-ary) input symbol alphabets is straightforward and is closely related to the radix-q extension of the Fast Fourier Transform [6]. Utilization of the processors and path metric storage locations in the cases of a q-ary alphabet will be $1/q$.

4 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 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 is given in [7]. The proof consists of four parts:

- Inter-stage Delay Computation — a general expression is derived for the minimum number of clock cycles between the time the first path metric is available from the j-th stage and the time the first computation in the $j+1$-st stage is able to begin.

- Local Time Computation — a general expression is derived for the number of clock cycles between the time the first path metric is available from the j-th stage and the time the path metric for a given state $a_{\nu-1}a_{\nu-2}\ldots a_2a_1a_0$ is available from the j-th stage.
- Acceptance Time Computation — a general expression is derived for the number of clock cycles between the time the first path metric is required at the inputs of the zeroth stage and the time the path metric of a particular state $a_{\nu-1}a_{\nu-2}\ldots a_2a_1a_0$ is required at the inputs of the zeroth stage.
- Causality Check — the Inter-Stage Delay and Local Time are added together to form Generation Time. We add $2^{\nu-1}$ (i.e. number of clock cycles required to complete all path metric computations of the previous stage) to the Acceptance Time to derive Consumption Time. We then verify that the Generation Time is no larger than the Consumption Time in order for causality to be maintained.

Figure 2(b) demonstrates how computations for the 16-state Viterbi Decoder can be performed on two processors instead of four, as illustrated in Figure 2(a). 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 CCVD and FCVD architectures (with ν and $\nu/2$ processors respectively), it is natural to ask whether other similar architectures exist with the number of processors that is not equal to ν or $\nu/2$. It turns out that it is indeed possible to extend the FCVD and the CCVD architectures into a family of architectures, known as *Generalized Cascade Viterbi Decoders* (GCVD) [7], with from one to $\nu-1$ processors that can be used to decode a single data stream with nearly 100% utilization and simultaneous deep pipelining of the ACS operation. When multiple interleaved data streams are decoded, the number of the processors that can be be used with 100% utilization is $\nu-1$ times the number of the interleaved data streams.

When decoding a single stream of data, the utilization of each processor will be [7]

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

Here, d is the pipelining depth of the ACS module, and k is the number of processors. Note that in addition to the primary linear speed-up region which runs from $k=1$ to $k=\nu-1$ there are also secondary, tertiary and so forth linear speed-up regions. As is apparent from Equation 1 these linear regions will have a slope that is inversely proportional to $1+\left\lfloor\frac{k}{\nu-1}\right\rfloor$, indicating poor utilization in decoding one stream of data. On the other hand, n interleaved streams of data($n \geq 1+\left\lfloor\frac{k}{\nu-1}\right\rfloor$) can be decoded with utilization that is very close to 100%.

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

For the same value of ν, increasing d will cause a decrease in utilization, however, increasing the pipelining depth d will permit an increase in a clock frequency and therefore also in the throughput. The optimum value of d for a given ν will depend on a number factors, including width (in bits) of the path metric and the arithmetic structure inside ACS.

5 INTERSTAGE SHIFT REGISTERS AND CROSS-POINT SWITCHES

Recall from our discussion in Section 3 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 j-th 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 re-order the path metrics before re-circulating them back to the inputs of the zeroth stage. We have stated in Section 3 that conceptually we can consider this re-ordering 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, instead we can perform the necessary re-ordering of the path metrics using a concatenation of the cross-point switches with associated shift registers [7] In general, a muti-stage network of cross-point switches is required. However, when k is equal to $\nu-1$ (or a multiple thereof), a much simpler network, consisting of just a single cross-point switch with two registers, is sufficient. In addition to the simplicity of a re-circulation network design, the GCVD with $\nu-1$ processors provides the maximum possible throughput increase.

6 IMPLEMENTATION OF THE GCVD

A $\nu=14$ GCVD with 13 processors, compatible with the JPL's proposed decoder for the Galileo space probe is under development. The decoder is intended to perform error correction on a deep space communication channel with a SNR of approximately 1dB. The code proposed by the Jet Propulsion Laboratory is a concatenated code, with an inner convolutional code and an outer Reed-Solomon code. The $(4,1,14)$ Viterbi decoder must be capable of decoding at a rate of 130 kb/s, with 8-bit soft quantization. A proposed 13-processor implementation will operate at clock frequency of 84.12 MHz. The pipelining depth of each ACS unit is assumed to be 17, with a pipeline register following each full adder in Add and Compare circuits as well as one pipeline register following the Select operation.

7 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. The proposed family of architectures is well suited for VLSI implementation.

A GCVD with $k=\nu-1$ Processors is especially attractive because it achieves the highest speed-up possible in decoding a single stream of data, with nearly 100% utilization and deep pipelining of the ACS datapath.

REFERENCES

[1] P. G. Gulak and T. Kailath. Locally Connected VLSI Architectures for the Viterbi Algorithm. *IEEE Journal on Selected Areas in Communications*, 6:527–538, April 1988.

[2] P. G. Gulak and E. Shwedyk. VLSI Structures for Viterbi Receivers: Part I – General Theory and Applications. *IEEE Journal on Selected Areas in Communications*, 4:142–154, January 1986.

[3] O. Collins, F. Pollara, S. Dolinar, and J. Statman. Wiring Viterbi Decoders (Splitting de Brujn Graphs). TDA Progress Report 42-96, Jet Propulsion Laboratory, Pasadena, California, October-December 1988.

[4] G. Fettweis and H. Meyr. Parallel Viterbi Algorithm Implementation: Breaking the ACS-Bottleneck. *IEEE Transactions on Communications*, 37:785–789, August 1989.

[5] G. D. Forney Jr. The Viterbi Algorithm. *Proc. IEEE*, 61:268–278, March 1973.

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

[7] G. Feygin. A Multiprocessor Architecture for Viterbi Decoders with Linear Speed-up. Master's thesis, University of Toronto, Toronto, Canada, 1990.

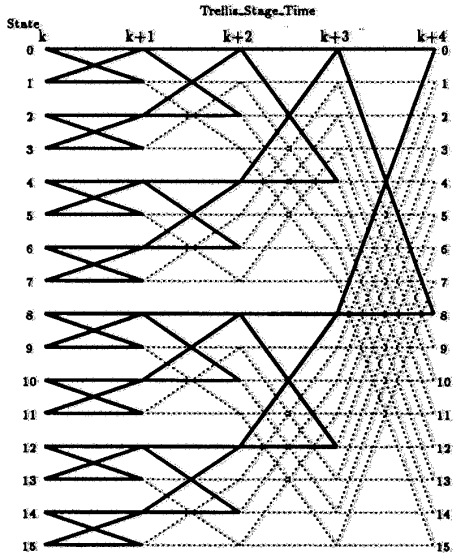

Figure 1. The dependance graph of the ACS operations over $\nu - 1$ stages of the trellis

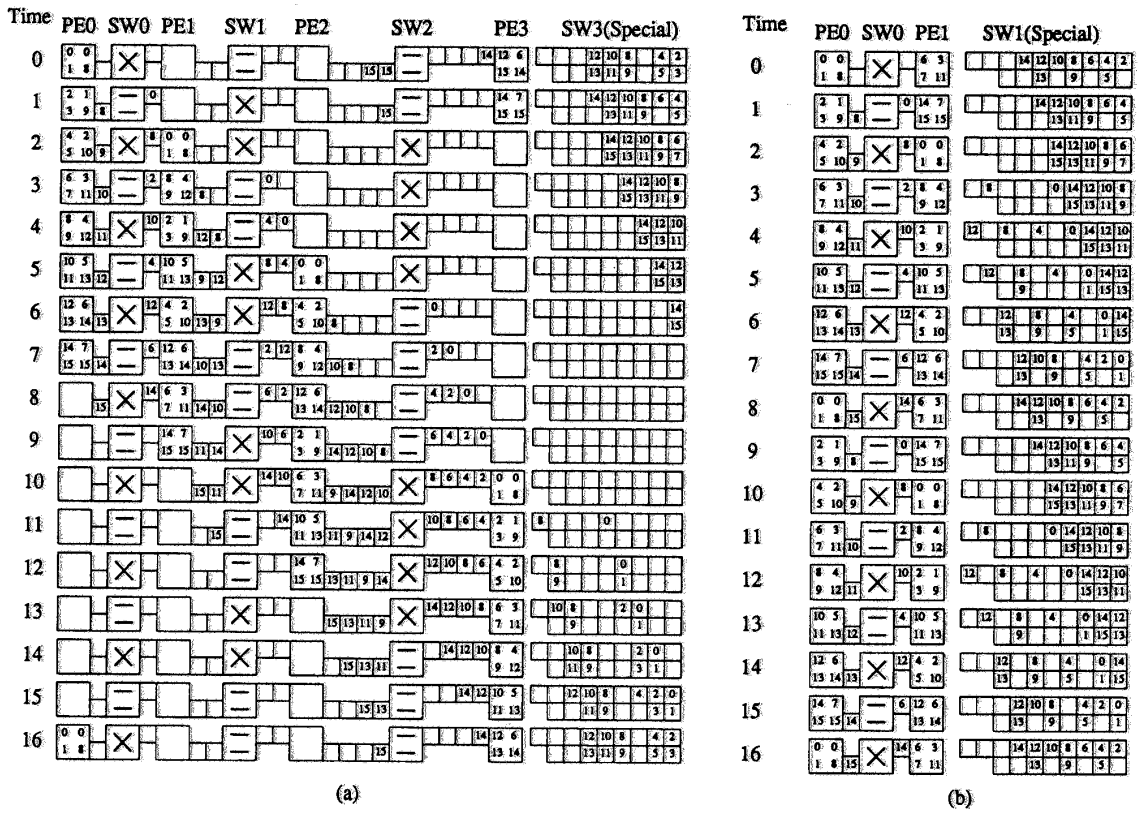

Figure 2. ACS schedule for a CCVD (a) and FCVD (b) Decoders, with $q = 2$, $\nu = 4$.