A Parallel Viterbi Decoding Algorithm

J.S. Reeve

Abstract

In this paper we express the Viterbi algorithm as a matrix-vector reduction in which multiplication is replaced by addition and addition by minimisation. The resulting algorithm is then readily parallelised in a form suitable for implementation on a systolic processor array. We describe the algorithm for BCH codes which have a task graph with valence restricted to four inputs and four outputs. The method is also applicable to convolution codes but the complexity of the task graph increases with the number of input bits for these codes. Results for BCH codes are given for two general purpose parallel machines, an IBM SP2 and a Meiko CS2.

Keywords

Viterbi decoding, parallel algorithms.

I. Introduction

THE Viterbi algorithm[1] was developed as an asymptotically optimal decoding algorithm for convolution codes. It is nowadays commonly also used for decoding block codes since the usual[2], [3] algebraic decoding methods are not always readily adaptable for soft decoding. In soft decision decoding the modem returns a measure of the relative probability that the data bit is a 0 or a 1. In these circumstances Viterbi decoding of Bose-Chaudhuri-Hocquenghem (BCH)[4], [5] and convolution codes codes is found to be efficient and robust for small codes. We illustrate the Viterbi algorithm for hard decision decoding (data bits are delivered as either 0 or 1) only, as the adaption to soft decision decoding is trivial.

Although the Viterbi algorithm is simple it requires $O(2^L)$ words of memory, where L is the length of the generating shift register in bits, which consequently has 2^L states. In practical situations it is desirable to select codes with the highest minimum Hamming distance that can be practicably decoded and an increased minimum Hamming distance d_{min} implies a longer shift register. Hence it is desirable to have a parallel Viterbi decoder that distributes the memory requirements among processors.

We describe our method for BCH codes as these are more complex than the convolution decoding, principally because of the presence of feedback in the generating shift register. The parallelisation strategy for convolution codes is the same as that for BCH codes.

II. The Sequential Viterbi Algorithm

BCH codes are a class of cyclic codes that append $n-k$ parity bits to a message of k bits so that each code word is n bits long. The code parameters (n, k, d_{min}) are of the form $n = 2^m - 1$, $n - k \leq mt$ and the minimum Hamming distance is $d_{min} \leq 2t + 1$. The codes are specified by their generator polynomials in $GF(2)$ which has the general form $g_0 + g_1 D + \cdots + g_{n-k} D^{n-k}$.

The encoding process is usually described in terms of a shift register. The general setup is shown in Figure 1 in which switches s_0 and s_1 are closed and s_2 open for the first k cycles while the message m_k of length k is input. For the next $n-k$ cycles switch s_2 is closed and switches s_0 and s_1 are open.

Although it is possible to program the functionality of the shift register encoder for a software simulation of the encoding process, it is more convenient to generate the state transition matrix which is necessary to implement the Viterbi decoder.

The state transition table has entries that are labelled by the state number - the base 10 number represented by the bit reversal of the shift register bits. The state transition table for the BCH code (7,4,1) with generator $D^3 + D + 1$ is shown in Table I, in which for example state 4 goes to state 3

J.S. Reeve The Department of Electronics and Computer Science University of Southampton, Southampton SO17 1BJ, UK. E-mail: jsr@ecs.soton.ac.uk

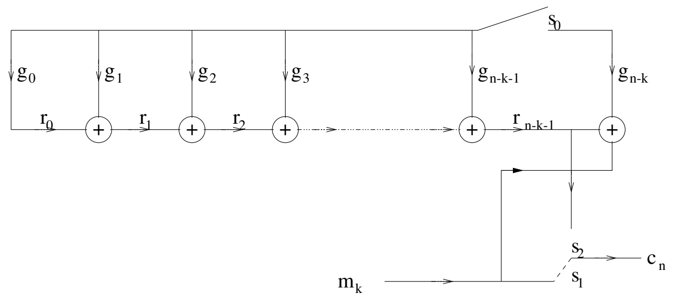

Fig. 1. Shift Register Encoding using BCH Codes

TABLE I
THE STATE TRANSITION TABLE FOR THE 7-4-3 BCH CODE

in_0	in_1	$State$	out_0	out_1
0	4	0	0	3
5	1	1	2	1
1	5	2	4	7
4	0	3	6	5
2	6	4	3	0
7	3	5	1	2
3	7	6	7	4
6	2	7	5	6

when a 0 bit is output and to state 0 when a 1 bit is output. Likewise state 4 is arrived at when state 2 outputs a 0 bit and also when state 6 outputs a 1 bit.

The sequential Viterbi decoding algorithm is best illustrated by example and by constructing the corresponding trellis diagram.

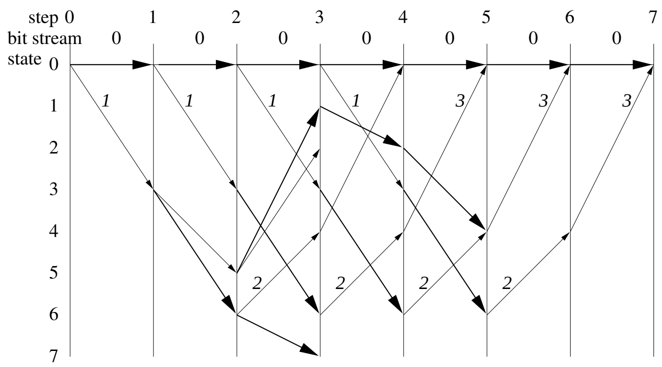

Fig. 2. The Viterbi Trellis for the BCH (7,4,3) Code with Input Sequence {0,0,0,0,0,0,0}

Figure 2 shows the decoding paths for the input sequence $\{0, 0, 0, 0, 0, 0, 0\}$. The state number on the left hand side represents the state of the encoding shift register (bit reversed with respect to Figure 1). The error corrected path is the one that starts and ends in state 0. Thick lines in the Figure 2 are the path branches for input bit 0 and thin lines are the path branches for the input bit 1. The weight of a path is its Hamming distance from the input stream, which in this case is simply the number of thin lines that it contains. Some of these weights are indicated by italic numbers on the diagram. Where more than one path meets at a node the one with the lowest weight is selected and the others discarded since these cannot result in complete paths with lesser weight. If paths at a node in the trellis diagram have equal weight then an arbitrary decision has to be made. At step $n-k$ there are 2^{n-k} active paths, one in each state and after step k the number of surviving paths is halved.

III. The Parallel Viterbi Algorithm

Although the trellis representation of the Viterbi algorithm is informative, it highlights the sequential nature of the algorithm. In this section we couch the Viterbi algorithm in terms of a path cost minimisation problem that is closely related to matrix multiplication. The parallel algorithm is then constructed by row-wise partitioning of this matrix. Our method is closely related to the parallelisation reported by Kumar [6] of Floyd's [7] minimum cost path algorithm.

We represent the state of the Viterbi trellis at a given time by a weight vector $\vec{w}$, each element, w_s, of which gives the Hamming weight of the current path in state s. The special value $w_s = d_{min}$ indicates that there is no correctable path through state s. As an example consider the BCH (7,4,3) code from the previous section. At the current time the weight of the path through each state is $w = (w_0, w_1, \cdots, w_7)$. To find the weight of the path through a given state at the next time step we look to the state table. For instance the state 0 can be reached from the state 0 if the data bit is 0 and from the state 4 if the input bit is 1. So the new value for the weight at the state 0 is

$$w_0' = \begin{cases} min\{w_0, w_4 + 1\} & \textit{on inputting a } 0 \\ min\{w_0 + 1, w_4\} & \textit{on inputting a } 1 \end{cases}$$

For decoding purposes it is convenient to represent this table as a matrix S in which the elements $s_{ij} = 1$ if state i can be arrived at from state j, otherwise $s_{ij} = 0$. The form of this matrix for the BCH (15,7,2) code is shown in Figure 3.

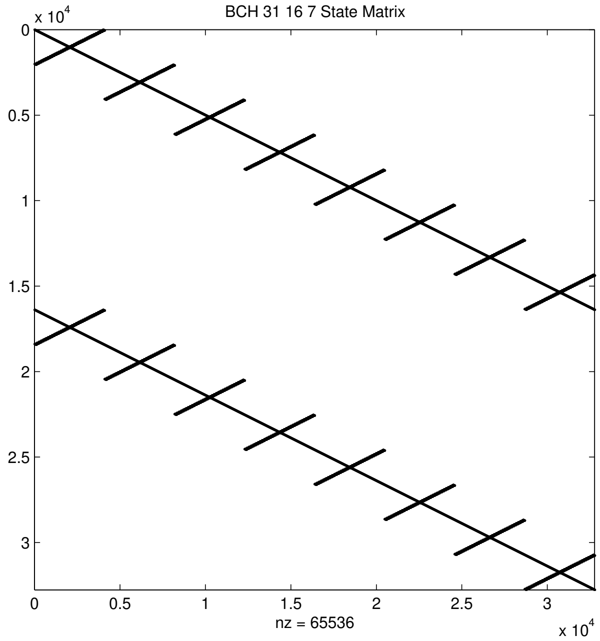

Fig. 3. The Form of the State Transition Matrix for the BCH (31,16,7) Code

Our parallelisation strategy is equivalent to distributing the S matrix in row-wise fashion so for the particular code shown in Figure 3 cutting the matrix in half results in a two processor solution for which the matrix-vector reduction is independently done on each processor although each processor requires all of the weight vector. If we continue to bisect the S matrix we generate more complex task graphs, so that for 8 processors the task graph looks like Figure 4 in which the circles represent processors and the arcs represent the transition of the paths and their Hamming weights, of those states whose row number of the S matrix is resident on the originating processor. This parallel decoding machine operates in lock step for n cycles for a (n, k, d_{min}) code. For BCH codes the in-valence of the task graph never exceeds 4, whereas for convolution codes the in-valence depends on the number, k, of input bits. The task graph for all $k = 1$ convolution codes of any constraint length on 8 processors is exactly that of Figure 4.

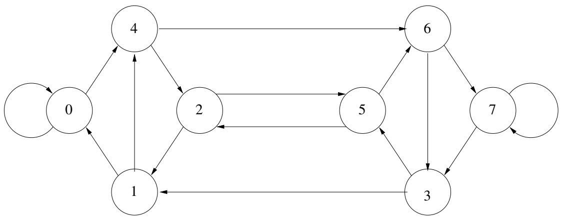

Fig. 4. Task Graph for the BCH (31,16,7) Code on 8 Processors

Thus our parallel Viterbi decoding algorithm for a (n, k, d_{min}) BCH code consists of n matrix multiplications each of which takes time proportional to n/p, where p is the number of processors. This gives an overall time complexity of $O(n^2/p)$. The memory complexity of our method is $O(2^{n-k})$ because the paths and their weights must be stored for each state. The S matrix does not need storing as it is efficiently generated as the algorithm proceeds.

IV. Results

We have timed our algorithm on two different general purpose parallel processors for a variety of codes whose sizes (number of states) and task graph valences are given in Table II.

Code	No of Procs			Size
(n, k, d_{min})	4	8	16	2^{n-k}
(255,239,5)	2	4	4	2^{16}
(63,45,7)	4	4	4	2^{18}
(31,11,5)	2	4	4	2^{20}
(127,106,7)	2	2	4	2^{21}

TABLE II

Task graph valences of the codes used in the evaluation. The size is the number of states for each code.

The first machine used was a Meiko CS2 which consists of 8 nodes each with two 125MHz SPARC II processors sharing 128 Mbytes of memory. These nodes are interconnected by a layered cross-bar switch of message latency $\approx 4 \times 10^{-4}$ seconds and an asymptotic bandwidth of 2 32-bit Mega-words per second when coding in C and using the MPI libraries[8] to handle the communications. The results for this machine are shown in Table III. There is insufficient memory on a single node to time the (127,106,7) code.

The other machine used was an IBM SP2 which consists of 16 nodes, each with a single 166MHz RS6000 processor and 256 Mbytes of memory, interconnected by a layered cross-bar switch of message latency $\approx 2.5 \times 10^{-5}$ seconds and an asymptotic bandwidth of 4 32-bit Mega-words per second again when coding in C and using the MPI libraries to handle the communications. The results for this machine are shown in Table IV

Code	Number of Processors				
(n, k, d_{min})	1	2	4	8	16
(255,239,5)	41	24	16	19	19
(63,45,7)	66	35	18	8	6
(31,11,5)	98	51	25	14	12
(127,106,7)	-	835	392	207	195

TABLE III
Timings (in seconds) to Decode Selected Codes on the CS2

Code	Number of Processors				
(n, k, d_{min})	1	2	4	8	16
(255,239,5)	26	29	19	18	15
(63,45,7)	13	12	11	6	4
(31,11,5)	22	18	11	9	6
(127,106,7)	259	267	174	87	86

TABLE IV
Timings (in seconds) to Decode Selected Codes on the SP2

V. Summary

Although from the results tables it is not always apparent that our parallel version of the Viterbi algorithm works efficiently, this is solely the effect of the communications costs. We have demonstrated this by running all the test cases without computing paths and weights but just passing the data. This gives timings that are always within 5% of the completely computed results. The origins of the communications costs lie in the large message latency and contention within the cross-bar switches. Our algorithm is clearly systolic and could be implemented very efficiently on a purpose built machine. We are currently embarking on a project to construct a reconfigurable Viterbi decoder using FPGA technology which will provide a machine architecture that directly maps the task graph for a particular code. This will circumvent the network contention problem and reduce the overhead of general purpose message passing software.

References

[1] A.J. Viterbi, "Error bounds for convolution codes and an asymptotically optimum decoding algorithm," *IEEE. Transactions on Information Theory*, vol. 13, pp. 260–269, 1967.

[2] E.R. Berlekamp, *Algebraic Coding Theory*, McGraw-Hill Inc, 1968.

[3] R.E. Blahut, *Theory and Practice of Error Control Codes.*, Addison-Wesley., 1983.

[4] R.C. Bose and D.K. Ray-Chaudhuri, "On a class of error-correcting binary group codes.," *Information and Control*, vol. 3, pp. 68–79, 1960.

[5] A. Hocquenghem, "Codes correcteurs d'erreurs," *Chiffres (paris)*, vol. 2, pp. 147–156, September 1959.

[6] V. Kumar, A. Grama, A. Gupta, and G. Karypis, *Introduction to Parallel Computing, Design and Analysis of Algorithms*, Benjamin-Cummings., 1994.

[7] R.W. Floyd, "Algorithm 97: Shortest path.," *Communications of the ACM*, vol. 5, no. 6, pp. 345, 1962.

[8] W. Gropp, E. Lusk, and A. Skjellum, *USING MPI, Portable Parallel Programming with the Message Passing Interface*, MIT Press., 1992.