A General Parallel Algorithm for State Machine Based Error Correction Decoding

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

Abstract

In this paper we express state machine based error correction decoding algorithms 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 dedicated hardware. As a demonstration of the versatility of the algorithm we apply it to the hard decision Viterbi decoding of BCH codes and to the soft decision decoding of Recursive Systematic Convolution (RSC) Codes of rate 1/2 for any constraint length N. Results obtained on an IBM SP2 machine using up to 16 processors for various minimum free distance RSC codes, and a variety of BCH codes are presented. Specifically for these codes , the resulting task graphs have valence restricted to four inputs and four outputs which makes the algorithm efficient and scalable.

1 Introduction

Ever since Shannon's[9] development of information theory and the realisation that real noisy communications channels can be made to behave as if they were perfect, there has been a concerted effort to find effective error correcting codes that allow the channel capacity to approach that of the Shannon limit. This is particularly important when bandwidth is expensive such as deep-space applications or mobile phones for instance. Sophisticated codes, however, do not always, or often, lend themselves to simple algebraic decoding methods. For this reason state transition machine based decoding algorithms, of which the Viterbi algorithm[10] is the best known example, have become very important. For short codes, which append few parity bits and consequently have limited error correction capability, extending the Viterbi trellis in time and allowing decoding to operate in pipeline manner is sufficient. However, when an extended error correction capability is required, the number of reachable states in such a decoding machine grows exponentially with the number of appended parity bits, and so direct implementation in hardware is not feasible.

In this paper we describe a method of breaking down the state space of the decoding state transition machine in such a way that each subspace interacts with only limited number of other subspaces, which makes this class of decoding algorithm amenable to parallelisation by domain decomposition. We demonstrate the effectiveness of our parallelisation method for hard decision decoding of Bose-Chaudhuri-Hocquenghem (BCH)[3, 6] codes and for the soft decision decoding of Recursive Systematic Convolution (RSC) codes. These latter codes are important as they are used in deep-space communications and are being muted for use in 4th generation mobile phones as the constituents of turbo encoding/decoding schemes [8, 1, 2]as they are readily adaptable for maximum likelihood soft decoding. In soft decision decoding the modem returns a number that represents the log of the probability ratio that the data bit is a 0 or a 1. In these circumstances state transition based decoding of RSC codes is found to be efficient and robust. Our test platform is a 16 processor IBM SP2 distributed memory machine.

2 The Sequential Algorithm

To demonstrate the principles of our parallelisation technique we use the exemplar of hard decision decoding of BCH codes using the 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 implemented as a particular wiring of a shift register, but for our purposes we need to describe it as a state transition diagram in which there are 2^{n-k} states.

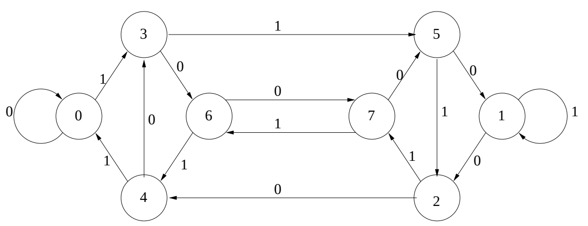

Figure 1: The State Transition Diagram for the BCH (7,4,3) Code

As an example the state transition diagram for the BCH(7,4,3) code is shown in Figure 1. Messages of 4 bits have 3 parity bits appended and the Hamming distance[1] between code words is 3. To encode the message $m_n = \{1, 1, 0, 1\}$ we start at state 0 and move to state 3, as the 1 input causes this transition according to the diagram. Proceeding in this way we arrive at the state 1. We must now return to state 0 in 3 transitions. This gives the parity bit sequence $p_n\{0, 0, 1\}$, and the transmitted sequence is $t_n = \{0 \xrightarrow{1} 3 \xrightarrow{1} 5 \xrightarrow{0} 1 \xrightarrow{1} 1 \xrightarrow{0} 2 \xrightarrow{0} 4 \xrightarrow{1} 0\}$

Now if we receive the sequence $r_n = \{0 \xrightarrow{1} 3 \xrightarrow{0} 6 \xrightarrow{0} 7 \xrightarrow{1} 6 \xrightarrow{0} 7 \xrightarrow{0} 5 \xrightarrow{1} 2\}$, which has one bit corrupted from the transmitted sequence, and follow it through the state transition diagram, we end in state 2 with Hamming weight 0. The sequence, $c_n = \{0 \xrightarrow{1} 3 \xrightarrow{1} 5 \xrightarrow{0} 1 \xrightarrow{1} 1 \xrightarrow{0} 2 \xrightarrow{0} 4 \xrightarrow{1} 0\}$, however, ends in state 0 with Hamming weight 1 and so the corrected message is $\{1, 1, 0, 1\}$. For this method to work we need to follow all possible paths through the state 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 there are two equally weighted paths through a node, one is selected at random. Paths with Hamming weight greater than t can also be discarded as not being correctable.

3 The Parallel Viterbi Algorithm

Although the state transition diagram 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 [7] of Floyd's [4] minimum cost path algorithm.

We represent the state of the Viterbi decoding process at a given time step by a weight vector $\vec{w}$, the elements 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 2.

Our parallelisation strategy is equivalent to distributing the S matrix in row-wise fashion so for the particular code shown in Figure 2 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 3 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 matrix. This parallel decoding machine operates in lock step for n cycles for

[1] The Hamming distance is the number of bits in which code words differ.

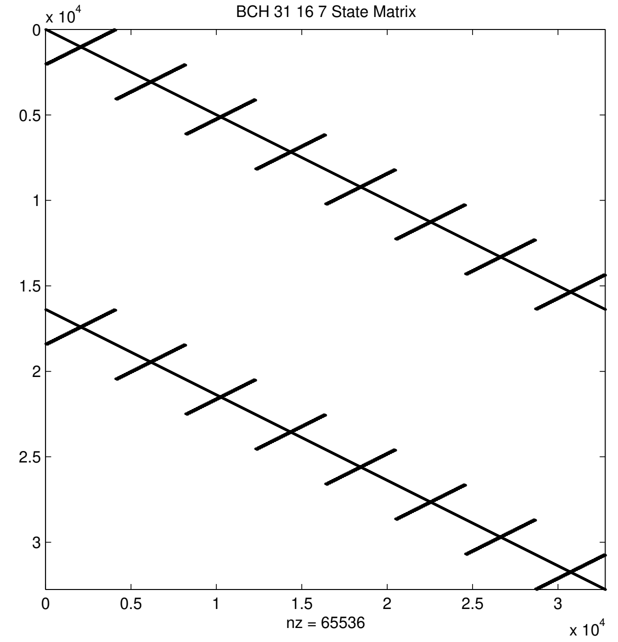

Figure 2: The Form of the State Transition Matrix for the BCH (31,16,7) Code

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 3.

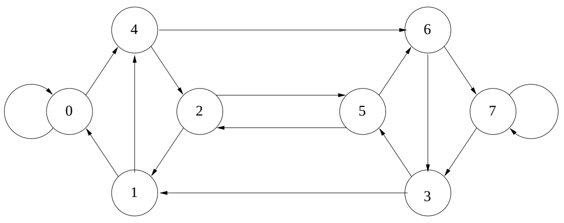

Figure 3: Task Graph for the BCH (31,16,7) Code on 8 Processors

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 of course doesn't need storing as it is efficiently generated as the algorithm proceeds.

For the soft decision decoding of RSC codes, the principle of the the method is precisely the same as that described for the hard decision Viterbi algorithm. Although the generation of the codes is more complex because of their recursive nature, the state space is decomposable in the same way, giving rise to very similar task graphs as we noted above. There is, however, a difference in the computational complexity of the decoding process. Rate 1/2 RSC codes are a subset of convolution codes that append 1 parity bits to every information bit so that each code word is $2n$ bits long for an n bit message frame. The code parameters (g_1, g_2, d_f, m) in which g_1 is a feedback polynomial, g_2 a feed forward polynomial (both usually expressed in octal form), the maximum free distance is d_f and m is the number of bits in the generating shift register. The encoding and decoding processes can also be expressed as state transition systems, for instance, the RSC code (15,17,6,3) has a state transition diagram topologically identical with Figure 1 and we can employ the same parallelisation strategy as described above. The additional computational complexity arises from the requirement that the log-likehood probabilities of each transition be estimated and then the selection decision made. For instance if the state 0 can be reached from the state 0 if the data bit is 0 and from the state 1 if the input bit is 1, then the new value for the weight at the state 0 is

$$w'_0 = min\{w_0 + p_0 y_0, w_1 + p_1 y_1\}$$

Depending on which of the valid parities p_0 or p_1 is used determines whether a 1 or 0 is appended to this most likely path. The y_0 and y_1 are the log-likehood values delivered by the modem. For these codes, the S matrix has only $s_{i,2iMOD2^m}$ and $s_{i,(2i+1)MOD2^m}$ as the only non zero elements.

Again, our parallelisation strategy is equivalent to bisecting S matrix in row-wise fashion. This parallel decoding machine operates in lock step for n cycles for a (g_1, g_2, d_f, m) code.

The memory complexity of our method is $O(2^m)$ because the paths and their weights must be stored for each state and again, the S matrix is generated as the algorithm proceeds.

4 Results

For each test case, the 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 when coding in C and using the MPI libraries[5] to handle the communications.

For the hard decision decoding of the BCH the structure of the task graph and the size of the state transition diagram is given in Table 1.

BCH 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 1: Task graph valences for the BCH codes used in the evaluation. The size is the number of states for each code.

The decoding times on the SP2 are given in Table 2.

BCH 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 2: Timings (in seconds) to Decode Selected BCH Codes on the SP2

For the RSC codes we have timed our algorithm for 100 frames of 100 bits each. The results for this machine are shown in Table 3.

RSC Code	Number of Processors				
(g_1, g_2, d_f, m)	1	2	4	8	16
(27123,21675,16,13)	222	151	91	58	31
(17661,10533,16,12)	115	87	58	31	16
(5723,4335,15,11)	52	43	29	16	8
(3661,2335,14,10)	24	20	13	7	5

Table 3: Timings (in seconds) to Decode Selected RSC Codes on the SP2

As these algorithms run in lock step we were able to set-up the communications pattern initially and then initiate all the data transfers with one *MPI_Startall* call.

5 Summary

Although from the results tables it is not always apparent that our parallel version for the hard decision decoding of BCH codes 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. This would circumvent the network contention problem and reduce the overheads of general purpose message passing software.

In the case of the RSC codes, from the results table it is apparent that our parallel version of the Viterbi algorithm works efficiently. Every time a bit is added to the generator register the state space doubles so times should double on a fixed number of processors, also the time on p processors for a code of memory length m should match that on $2p$ processors for a code of memory length $m + 1$. This ideal scenario is not quite borne out by our experiments but again this is solely the effect of the communications costs and again purpose built hardware would alleviate this.

References

[1] A. Bahl, J. Cocke, F. Jelinek, and J. Raviv. Optimal decoding of linear codes for minimumizing symbol error rate. *IEEE Transactions in Information Theory*, pages 284–287, March 1974.

[2] C. Berrou, A. Glavieux, and P. Thitimajshima. Near shannon limit error correcting coding and decoding: Turbo codes. In *Proc. Int. Conference on Communications*, pages 1064–1070, 1993.

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

[4] R.W. Floyd. Algorithm 97: Shortest path. *Communications of the ACM*, 5(6):345, 1962.

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

[6] A. Hocquenghem. Codes correcteurs d'erreurs. *Chiffres (paris)*, 2:147–156, September 1959.

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

[8] W.E. Ryan. A turbo code tutorial. http://www.ece.arizona.edu/˜ ryan/.

[9] C.E. Shannon. A mathematical theory of communication. *Bell Systems Technical Journal*, 27:379–423, 1948.

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