An Efficient Architecture for High Speed Turbo Decoders[1]

Aliazam Abbasfar and Kung Yao

Department of Electrical Engineering,
University of California Los Angeles

ABSTRACT

Turbo codes not only achieve near Shannon-capacity performance, but also have decoders with modest complexity, which is crucial for implementation. So far efficient architectures for decoding of turbo codes have been proposed that is suitable for sequential processing. In this paper a novel parallel processing architecture for very high-speed turbo decoder is presented. The performance of this decoder and the tradeoff between speed and efficiency are discussed. It is shown that some decoders can run faster by some orders of magnitude while maintaining almost the same processing load. A systolic implementation of this decoder is presented at the end[1].

1. INTRODUCTION

Recently, some new classes of channel encoders have been introduced that achieve near Shannon-capacity rates. Turbo codes [4] and Low Density Parity Codes (LDPC) are the most important examples. The basic property of these codes is the capability of iterative decoding. The iterative algorithms can be viewed as a probability or “belief” propagation algorithm, which is based on message passing [1].

Although iterative decoding has a parallel nature for LDPC, for turbo codes it is very attractive as a sequential processor because messages can be computed iteratively using previously computed messages. Although the message passing algorithm can be parallelized in theory, it is quite inefficient and impractical for implementation. In [2] a concurrent turbo decoder is studied. Although the concurrent decoder can run by some orders of magnitude faster than the sequential counterpart, the number of components used for processing is so large that makes it quite impractical. Moreover, the processing load has been increased dramatically, which translates to low efficiency. Another approach proposed in the literature is using overlapping windows [3]. However, for a very high-speed decoder the extra processing load for overlapping bits causes inefficiency and irregularity.

In this paper we will propose a method that make parallel processing feasible, while efficiency of the decoder is maintained. Regular structures such as systolic arrays are proposed for implementation of such a decoder. In section 2 we will describe the decoding algorithm for turbo decoders. In section 3 the proposed method is described and the tradeoff between speed and efficiency is discussed. The performance of the proposed decoder is discussed in section 4. Finally, in section 5 a systolic array implementation of the decoder is presented.

2. TURBO CODE

Turbo code was introduced in [4]. Berrou, et al. presented the Parallel Concatenated Convolutional Code, (PCCC) and the iterative decoding algorithm. Later Serial Concatenated Convolutional Codes (SCCC) was presented. PCCC has been remained the most popular type of turbo code, which has been adopted in UMTS standards as channel coding scheme. In the following we briefly describe the PCCC encoder and its iterative decoder.

A PCCC is constructed from 2 or more parallel convolutional encoders that are working on the input sequence and the permuted version of it in parallel. Each convolutional code is called a constituent code. Figure 1 depicts the structure of a PCCC with two constituent codes. The block denoted by I is the interleaver, which permutes the input sequence with a predefined random pattern.

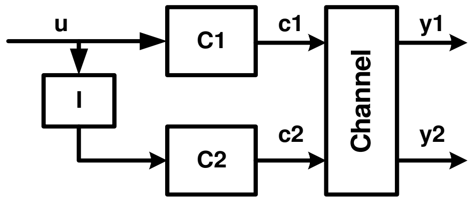

Figure 1: The structure of a PCCC encoder

The iterative decoding algorithm is based on Maximum-A-Posteriori (MAP) decision of the input sequence. However, since it is difficult to find the MAP

[1] This work is partially supported by a UC CoRe grant sponsored by ST Microelectronics, Inc. and NASA-Dryden grant NCC2-374.

solution by considering all the observations at the same time, the MAP decoding is performed on the observations of each constituent code separately. Since two codes have been produced from one input sequence, the A-Posteriori-Probability (APP) of data bits coming from the first decoder can be used by the second decoder and vice versa. Therefore the decoding process is carried out iteratively. In [5] a general unit, called SISO, is introduced that generates the APPs in the most general case.

Since the second constituent code is using the permuted version of the input sequence, therefore, extrinsic information also should be permuted before being used by the second decoder. Likewise, the extrinsic information of the second decoder is to be permuted in reverse order for the next iteration of the first decoder. Figure 2 shows the iterative decoding bock diagram.

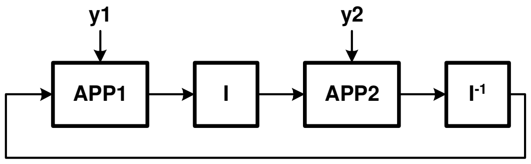

Figure 2: The iterative decoding block diagram

An efficient algorithm for MAP decoding of a convolutional code is known as BCJR algorithm [5]. In this algorithm A-Posteriori-Probabilities for a time-invariant trellis encoder can be computed with a complexity that depends linearly on the number of states and also on the size of input sequence. It should be noted that SISO is a block which implements the BCJR algorithm. Here we briefly describe the structure of this algorithm. For more details see [5-6]. The main three steps of this algorithm are as follow:

Forward recursion: In this step we compute the likelihood of all the states in the trellis given the past observations. Starting from a known state, we will go ahead along the trellis and compute the likelihood of all the states in one trellis section from the likelihood of the states in the previous trellis section. This iterative scheme is continued until likelihoods of all the states, which are called alpha variables, are computed in the forward direction.

Backward recursion: This step is quite similar to the forward recursion. Starting from a known state at the end of the block, we compute the likelihood of previous states in one trellis section. Therefore we compute the likelihood of all the states in the trellis given the future observations, which are called beta variables. This iterative processing is continued until the beginning of the trellis.

Output computation: Once the forward and backward likelihoods of the states are computed, the extrinsic information can be computed from them. The extrinsic information can be viewed as the marginal probability of each bit given the observations.

3. PARALLE TURBO DECODER

In this section we present a novel method for iteratively decoding the turbo codes. Although this method is applicable for every turbo code, we will explain it in the case of a block PCCC code. To obtain block codes, termination or tail-biting methods is used.

The algorithm is as following. First of all, the received data for each constituent codes are divided into several contiguous non-overlapping sub-blocks; so called windows. Then, each window is decoded separately in parallel using the BCJR algorithm. In other words, each window is a vector decoder. However, the initial values for alpha and beta variables come from previous iteration of adjacent windows. Since all the windows are being processed at the same time, in the next iteration the initial values are ready to load. Therefore, there is no extra processing needed for the initialization of state probabilities at each iteration. The size of windows is a very important parameter that will be discussed later. Figure 3 shows the structure of the decoder.

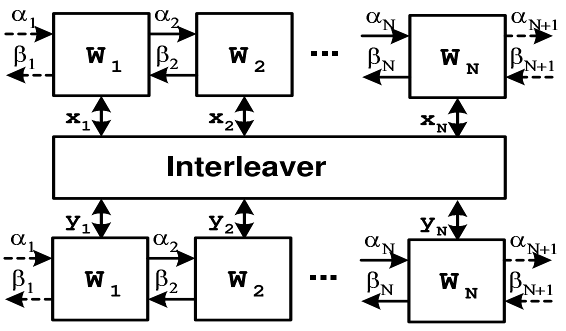

Figure 3: Parallel turbo decoder structure

The proposed structure stems from the message-passing algorithm itself. We have only introduced some new messages that are passed between sub-blocks at each iteration. There are two types of messages that are communicated between sub-blocks. First, the messages associated with the decoded data are the same as extrinsic information, which are communicated between two constituent codes in the traditional approach. Second, the messages that are related to the states in window boundaries, we call them state messages. These messages are the same as alpha and beta variables that are computed in forward and backward recursion of the BCJR algorithm. In the first iteration there is no prior knowledge available about the state probabilities. Therefore the messages are set to ½ for unknown states. In each iteration, these messages are updated and passed across the border of adjacent windows.

The optimum way to process a window is the sequential processing using forward and backward recursions; i.e. BCJR algorithm. Therefore each window processor is a SISO.

The processing of the windows in two constituent codes can be run in parallel. However, since this scheme can be exploited in the sequential decoder as well, this is not considered here for a fair comparison. In other words, that parallelization introduces another speed gain factor which can be exploited. Therefore the architecture of the decoder of the choice only needs half of the processors as it is shown in figure 4.

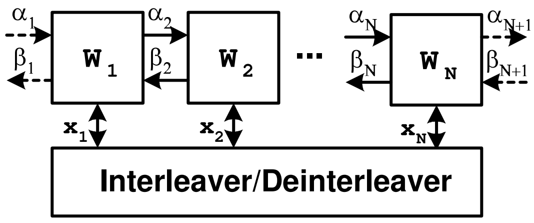

Figure 4: Parallel turbo decoder with shared processors for two constituent codes

Table I shows the parameters of a decoder. For window size at two extremes, the approach is reduces to known methods. If window size is B, the number of windows is 1, it turns out to the sequential approach. If the window size is 1, the architecture reduces to what was proposed in [2]. It should be noted that the memory requirement for all cases is the same.

Parameter	Definition
W	Window size
N	Number of windows
B = W x N	Block size
I	Number of iterations
T_W	Window Processing Time
$T = 2I\text{x } T_W$	Processing Time (Latency)
P = k 2I B	Processing Load

Table I: the decoder parameters

Two characteristic factors should be studied as performance figures. One is the speed gain and the other is the efficiency. These two are defined as following:

Speed gain $= T_0/T = N \text{ x } I_0/I$

Efficiency $= P_0/P = I_0/I$

Where T_0 and P_0 are the processing time and processing load for the sequential approach, i.e. W=B case.

This is very interesting result. The speed gain and the efficiency are proportional to the iteration ratio. If the number of iterations required for the parallel case is the same as the serial case, we enjoy a speed gain of N without losing the efficiency, which is ideal parallelization. Therefore we should look at the number of iterations required for a certain performance to further quantify the characteristic factors. In next section we will illustrate the performance of the proposed architecture for some widow sizes.

4. Simulation results

For simulations a PCCC with block size of 4800 is chosen. The interleaver is an S-random interleaver. The first constituent code is a rate one-half systematic code and the second code is a rate one non-systematic recursive code. The feed forward and feedback polynomials are the same for both codes and are $1+D+D^3$ and $1+D^2+D^3$ respectively. Thus coding rate is 1/3. The simulated channel is an AWGN channel.

The bit error rate performance of the proposed decoder has been simulated for window sizes of 64, 48, 32, 16, 8, 4, 2, and 1. The maximum number of iterations for each case is chosen such that the BER performance of the decoder equals that of the sequential decoder after 10 iterations. This is very important that this structure does not sacrifice performance for speed. We can always increase the maximum number of iterations to get similar performance as of the sequential decoder.

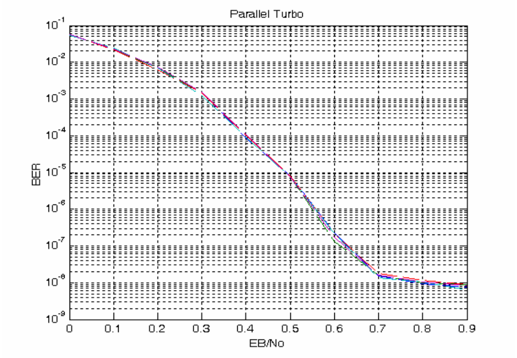

Figure 5: Performances of parallel decoder

However, in practice, the iterations are stopped based on a criterion that shows the data is reliable or correct. We have simulated such a stopping criterion in order to get the average number of iterations needed. The stopping rule that we use is the equality between the results of two consecutive iterations. The average number of iterations is used for the efficiency computation. The average number of iterations for low signal to noise ratio is the maximum number of iterations for each window size. Figure 5 shows the BER performance of the decoders. The curves are almost indistinguishable.

Efficiency of the parallel decoder with different window sizes is shown in Figure 6. It clearly shows that we have to pay some penalty in order to achieve speed gain. Also we observe that the efficiency of parallel decoder decreases gracefully for window sizes greater than 32. The efficiency is degraded dramatically for very small windows, which prohibits us to get speed gain as well. Another interesting thing in the efficiency curves is the flatness of the curves. In other words, the efficiency of the parallel decoder is almost constant in all SNR. This observation translates to almost constant speed gain over the whole SNR range.

As a summary, in Table II the maximum number of iterations, the average number of iterations, and the characteristic factors are tabulated for different window sizes at Eb/N0 = 0.7 (BER = 1e-8).

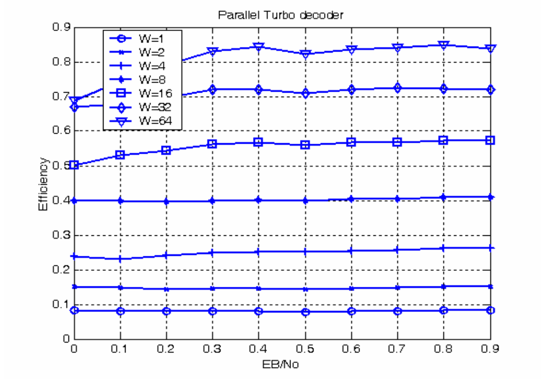

Figure 6: Ratio of the average number of iterations in parallel decoder to

Window Size	Max # of iterations	Ave. # of iterations	Speed Gain	Efficiency %
64	12	5.0	63	84
32	14	5.8	109	72
16	18	7.4	170	57
8	25	10.4	242	40
4	42	16.3	310	26
2	65	28.3	356	15
1	120	52.0	386	8

Table II: Characteristic factors for the parallel decoder at SNR= 0.7 dB (BER=10e-8)

5. Systolic array implementation

Systolic architectures for signal processing algorithms including Viterbi decoding have been proposed [7-9]. The regularity of the architecture presented for parallel turbo decoder suggests that we can implement it with systolic array type hardware. The design consists of several window processors (SISO) that decode a vector of data bits as shown in figure 4. The corresponding observations related to each vector are stored in its SISO. The SISO uses the observations, the initial alpha and beta coming from adjacent windows, and extrinsic information to produce new extrinsic information and new alpha and beta. This procedure repeats by the number of iteration by the same hardware. The other alternative is to do each iteration in a pipelined fashion. In this method the speed will be increased further. However, the memory requirement is much larger in this case. Achieving higher speed is better served by increasing number of windows because there is no memory penalty for this. It should be noted that a higher speed is achieved with a larger block size at the penalty of raising the memory requirement.

6. CONCLUSIONS

We have proposed an efficient architecture for parallel implementation of turbo decoders. The advantage of this architecture is that the increase in the processing load due to parallelization is minimal. Simulation results demonstrate that this structure not only can achieve some orders of magnitude in speed gain, but also maintains the efficiency in processing. Also we have shown that the efficiency and the speed gain of this architecture are almost independent of the SNR.

The regularity of the proposed architecture is another advantage. Therefore it is very suitable for VLSI implementation. One realization of such hardware in the form of systolic arrays was presented.

7. REFERENCES

[1] F.R. Kschischang and B.J. Frey, “Iterative dec. of compound codes by probability propagation in graphical models,” IEEE JSAC, pp. 219-230, Feb. 98.

[2] Frey, B.J.; Kschischang, F.R.; Gulak, P.G. “Concurrent turbo-decoding,” Proc. of IEEE International Symp. on Info. Theory, p. 431. July 97.

[3] J. Hsu and C.H. Wang, “A parallel decoding scheme for turbo codes,” Proc. ISCAS’98, vol.4, June 1998, pp. 445-448.

[4] C. Berrou, A. Glavieux, and P. Thitimasjshima, “Near Shannon limit error correcting coding and dec.: Turbo codes (1),” Proc. IEEE ICC., May 1993, pp. 1064-1070.

[5] S. Benedetto, D. Divsalar, G. Montorsi, and F. Pollara, “Soft-input Soft-output APP module for iter. decoding of conccat. codes,” IEEE Commu. Letters, pp.22-24, Jan. 97.

[6] L.R. Bahl, J. Cocke, F. Jelinek,, and J. Raviv, “Optimal dec. of linear codes for min. symbol error rate,” IEEE Trans. Inform. Theory, pp. 284-287, Mar. 1974.

[7] C.Y. Chang and K. Yao, “Systolic array processing of the Viterbi algorithm,” IEEE Trans. Inform. Theory, pp. 76-86, Jan. 1989

[8] G. Fettweis and H. Meyr, “High-speed parallel Viterbi decoding: Algorithm and VLSI architecture,” IEEE Commun. Mag., pp. 46-55, May 1991.

[9] F. Daneshgaran and K. Yao, “The iterative collapse algorithm: A novel approach to the design of long constraint length Viterbi decoders – Part I,” IEEE Trans. on Commun., pp. 1409-1418, Feb. 1995.