A REAL-TIME EMBEDDED SOFTWARE IMPLEMENTATION OF A TURBO ENCODER AND SOFT OUTPUT VITERBI ALGORITHM BASED TURBO DECODER

M. Farooq Sabir, Rashmi Tripathi, Brian L. Evans and Alan C. Bovik

Dept. of Electrical and Comp. Eng., The University of Texas, Austin, TX, 78712-1084 USA
{mfsabir,rashmi,bevans,bovik}@ece.utexas.edu

ABSTRACT

Turbo codes are used for error protection, esp. in wireless systems. A turbo encoder consists of two recursive systematic convolutional component encoders connected in parallel and separated by a random interleaver. A turbo decoder, which is iterative, is typically based on either a Soft Output Viterbi Algorithm (SOVA) or a maximum a posteri (MAP) algorithm. MAP is roughly three times more computationally complex than SOVA, but provides 0.5 dB of coding gain. In this paper, we implement a turbo encoder and SOVA-based turbo decoder in real-time software on a TMS320C6700 digital signal processor (DSP). The contributions of this paper are: (1) first publicly available implementation of a SOVA-based turbo decoder on a C6000 DSP, (2) speedup of 162x for the encoder on a C6200 DSP and 11.7x for the decoder on a C6700 DSP over level three C compiler optimization, and (3) dataflow modeling for a turbo channel coding subsystem.

1. INTRODUCTION

Turbo codes [1] can be used for error protection (e.g., in the cdma2000 standard) and in joint source-channel coding. An example of joint source-channel coding is the use of turbo codes in conjunction with JPEG2000 for image communication over noisy channels [2]. The error protection can be unequal in that some message bits receive more protection than others. Unequal error protection is particularly useful in scalable bitstream image transmission in which the most important visual information is encoded first. In embedded foveation image coding (EFIC) [3], for example, wavelet coefficients that contribute more to the foveated visual quality are encoded and transmitted first. This EFIC bitstream can be truncated at any point to give a higher compression ratio as compared to that of the original image.

In this paper, we present a real-time implementation of a rate compatible punctured turbo encoder and an iterative soft output viterbi algorithm (SOVA) based turbo decoder on a TMS320C6000 digital signal processor (DSP). This is a first publicly available implementation of a SOVA based turbo decoder on a C6700 DSP processor. After code and memory optimizations, a speedup of 162x for the encoder and 11.7x for the decoder were achieved, over level three C compiler optimization. Dataflow modeling for this turbo encoding and decoding system is also presented in this paper. The source code described in this paper is available online at $http://live.ece.utexas.edu/research/turbodsp/index.htm$

B. L. Evans was supported by The State of Texas Advanced Technology Program under project 003658-0614-2001.

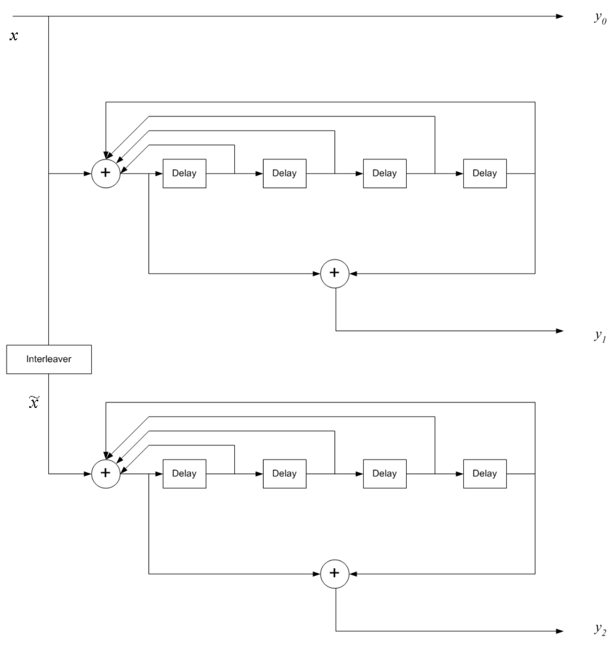


Fig. 1. A Rate $1/3$ Turbo Encoder [1]

2. TURBO CODES

A turbo encoder consists of two "recursive systematic convolutional" encoders, connected in parallel [1, 4, 5]. The input to the second encoder is an interleaved version of the input to the first encoder. This structure is called parallel because the input to both of the encoders is the same set of bits, rather than the output of one being the input to the other, as in serial concatenated codes. The component codes in

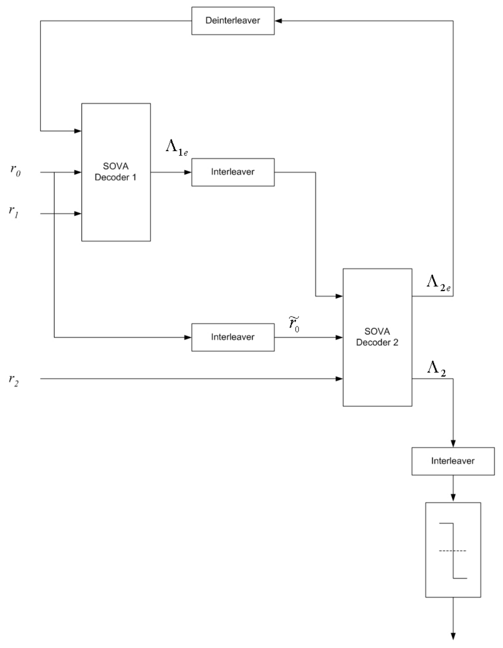


Fig. 2. Iterative SOVA based Turbo Decoder [1]

a turbo encoder are two recursive systematic convolutional encoders. The are called "systematic" because one of the outputs of the encoders is the input itself, whereas the property "recursive" comes due to the presence of a feedback loop in the encoders. A rate $1/3$ turbo encoder is shown in Fig. 1.

The generator matrix of a rate $1/2$ component code can be represented as

$$G(D) = \begin{bmatrix} 1 & \dfrac{g_1(D)}{g_0(D)} \end{bmatrix} \qquad (1)$$

where D represents the delay and $g_0(D)$ and $g_1(D)$ are the feedback and feedforward polynomials respectively. The degree of these polynomials is n, which depends on the number of delays in the convolutional encoders. As shown in Fig. 1, the same set of input bits is encoded twice, once by each component encoder. The input to the first encoder is the original input sequence x, whereas the input to the second encoder is the interleaved version of x, denoted by $\widetilde{x}$. The first output of the turbo encoder, which is the first output of the first encoder as well, is the original input sequence itself, denoted by y_0. The second output of the first encoder is the parity information y_1. For the second encoder we are only concerned with the parity output y_2 and not with the systematic bits. Hence the three outputs y_0, y_1 and y_2 are multiplexed to form the resultant output of the turbo encoder. Hence the resulting rate of this turbo encoder is $1/3$.

2.1. Interleaving for turbo codes

The interleaver used in the turbo codes is a pseudo-random block scrambler, defined by a permutation of N elements with no repetitions [1]. In this interleaver, a block of N inputs are read into the interleaver, and the output is read pseudo-randomly. Hence the interleaver plays two important roles in turbo encoders. On one hand it generates long block codes by using small memory convolutional codes, and hence achieves a coding gain, whereas on the other hand it decorrelates the inputs to the two encoders, so that an iterative suboptimum decoding algorithm can be applied, based on the information exchange between the two encoders. Due to the use of interleaver, there is a high probability that all the three output bits corresponding to an input bit are not corrupted at the same time (burst error), and hence after correcting some errors in the first decoder, a few more errors can be corrected in the second decoder. This process can be carried out iteratively, exchanging information between the two decoders again and again, and hence correcting more errors in every iteration. The interleaving must be available at the decoders as well, in order for the decoding to be performed correctly.

2.2. Puncturing

After encoding, puncturing is performed on the encoded bitstream. In order to provide unequal error protection to the encoded bitstream, different levels of puncturing are applied at different parts of the encoded bitstream. The puncturing is employed such that the higher rate codes are embedded in the lower rate codes. The codes generated by this scheme of puncturing are called rate compatible punctured codes. In this paper we employed rate compatible punctured turbo codes to provide different levels of error protection to the different parts of the bitstream. All the higher level codes were derived from the same rate $1/3$ mother code, produced by the encoder shown in Fig. 1.

2.3. Turbo Decoding

The decoding of the turbo codes can be performed using the Maximum a Posteriori (MAP) algorithm or the Maximum Likelihood (ML) algorithm based on the overall trellis of the code. However, these methods can be implemented only for short interleavers and are too complex for medium and long interleavers. One of the important practical features of turbo codes is the use of a simple suboptimum algorithm for decoding. One important reason why simple MAP decoding is not practical is that the overall trellis for the turbo codes is time varying and the number of states grows exponentially with the size of the interleaver in the turbo codes. Hence the MAP algorithm can only be used to decode in the case of very short interleavers. Due to these reasons,

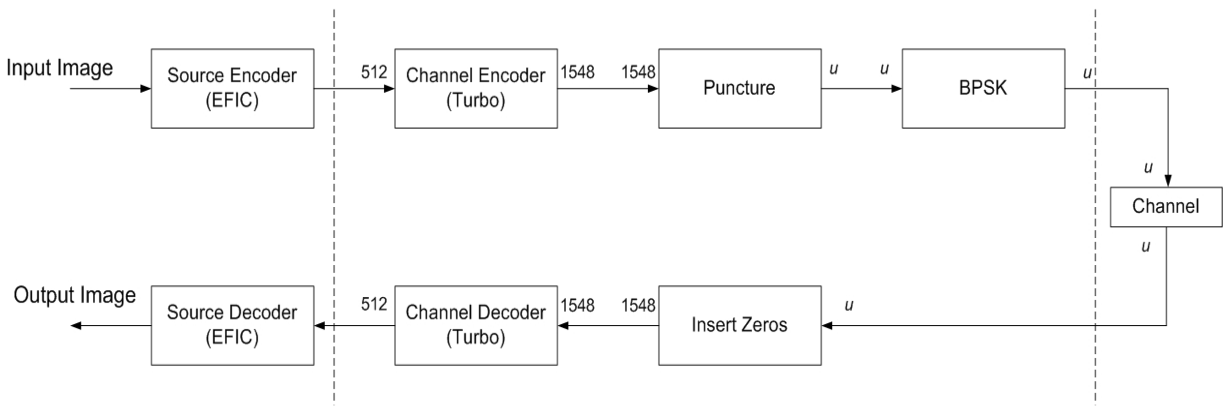


Fig. 3. Image transmission system with unequal error protection using an embedded foveation image coding (EFIC) encoder/decoder for the source coding and turbo encoder/decoder for the channel coding. The numbers above input (output) ports mean the number of bits input (output) per invocation of the block. BPSK means binary phase shift keying, i.e. two-level modulation.

iterative decoding algorithms are used to decode the turbo codes. Different iterative decoding algorithms like the Iterative MAP [1], the Iterative Log-Map [1] and the Iterative Soft Output Viterbi Algorithm (SOVA) [1] are used to decode the turbo codes. In this report we only present iterative SOVA because it is shown in [1, 6] that the complexity of the iterative SOVA is much lower than the MAP and the Log-MAP algorithms.

2.3.1. Decoding of turbo codes using iterative SOVA

The block diagram of the iterative SOVA is shown in Fig. 2. The input to the first decoder is the received bits corresponding to the actual information (systematic) bits, and the output of the first component encoder, represented by r_0 and r_1 respectively. This first decoder generates a soft estimate of the output and the extrinsic information Λ_{1e}. This extrinsic information is passed through the same interleaver as the one used in the encoder and is passed to the second decoder, to be used as an estimate of the priori probability by the second decoder. The inputs to the second decoder are the received bits corresponding to the output of the second encoder, represented by r_2, and r_0 passed through the same interleaver as used at the encoder, represented by $\widetilde{r_0}$. The second decoder also generates a soft decision Λ_2 and the extrinsic information Λ_{2e}. This extrinsic information is then passed back to the first encoder after deinterleaving, as an a priori estimate for the next decoding iteration. The decoder generates a hard decision after the desired number of iterations I are performed and then deinterleaves it and passes it to the output.

3. DATAFLOW MODELING

In this section we discuss how the different blocks of our system were modelled using different models of computational dataflow. Fig. 3 shows the number of input and output samples for the different blocks in the channel coding subsystem and channel model. The bitstream is divided into 512 sub-blocks of size 512 samples for ease of implementation. Different groups of these sub-blocks were then protected using different rate codes. The different values of the number of output samples from the channel coding subsystem, u, is shown in Fig. 3. The values of u depend on the part of the bitstream being encoded, i.e. the block number being processed, and are shown in Fig. 4. The values of u correspond for different puncturing vectors ranging from rate $1/3$ to rate $1/1$ punctured turbo codes.

The channel coding subsystem in Fig. 3 is periodic. For system-level modeling and simulation of periodic systems, two common dataflow models are synchronous dataflow and cyclostatic dataflow [7]. In synchronous dataflow, each block consumes (produces) a fixed positive integer number of samples on each input (output) port, and this behavior does not change during execution. In cyclostatic dataflow, the consumption (production) of samples on a port has fixed but periodic behavior. Cyclostatic dataflow is a superset of synchronous dataflow.

The turbo encoder, turbo decoder, and the binary phase shift keying (BPSK) modulator are naturally modelled using synchronous dataflow. The puncture and the insert zeros blocks are naturally modelled in cyclostatic dataflow, although they can also be modelled in synchronous dataflow when using a long enough period. Synchronous dataflow and cyclostatic dataflow graphs have the huge advantage

Block Number	Input Samples	Output Samples
1–4	1548	1548
5–8	1548	1420
9–16	1548	1292
17–32	1548	1162
33–64	1548	1032
65–128	1548	839
129–256	1548	645
257–512	1548	516

(a) puncture block

Block Number	Input Samples	Output Samples
1–4	1548	1548
5–8	1420	1548
9–16	1292	1548
17–32	1162	1548
33–64	1032	1548
65–128	839	1548
129–256	645	1548
257–512	516	1548

(b) insert zeros block

Fig. 4. Number of input and output tokens for each block. Behavior is cyclic, and repeats every 262144 samples (512 blocks $\times$ 512 samples/block).

that they can always be statically scheduled. Static schedules are useful for fast simulation and efficient synthesis. Synopsys Co-Centric System Design Studio [7] supports cyclostatic dataflow, and Agilent's Advanced Design System supports synchronous dataflow.

4. DSP SOFTWARE IMPLEMENTATION

We implement the channel coding subsystem in Fig. 3 on a TMS320C6701 DSP. A SOVA-based turbo decoder has not been publicly released on this DSP family before, although a MAP-based turbo decoder has been [8]. Our goal is to optimize the turbo encoder and SOVA-based turbo decoder for computation time. It will be critical to have all code and critical data segments reside on-chip. Since large blocks of data are needed to be stored efficiently in the internal memory for every subsequent iteration, limited on-chip data memory size of 64KB imposes a challenging task on the optimization of highly complex encoder and iterative SOVA decoder.

We first wrote all of the modules of the channel coding system in C and then compiled them on the TMS320C6701 DSP using Code Composer version 1.0. Ultimately, we implemented the encoder, puncture block, and BPSK modulator fixed-point assembly. We realized the SOVA-based decoder and the insert zeros block in floating-point assembly.

Optimization Stage	Cycle Count (Millions)
Original Code	4.18
Level 3	3.07
Memory Optimization	1.20
Loop Unrolling	0.135
Assembly	0.019

(a) encoder

Optimization Stage	Cycle Count (Millions)
Original Code	170
Level 3	119
Memory Optimization	21.1
Loop Unrolling	18.3
Assembly	10.2

(b) decoder

Optimization Stage	Cycle Count (K)
Original Code	128.7
Level 3	97.3
Memory Optimization	20.1

(c) puncture block

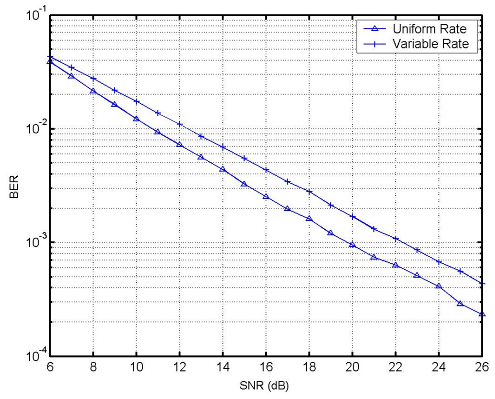
Optimization Stage	Cycle Count (K)
Original Code	821.4
Level 3	659.1
Memory Optimization	505.6

(d) insert zeros block

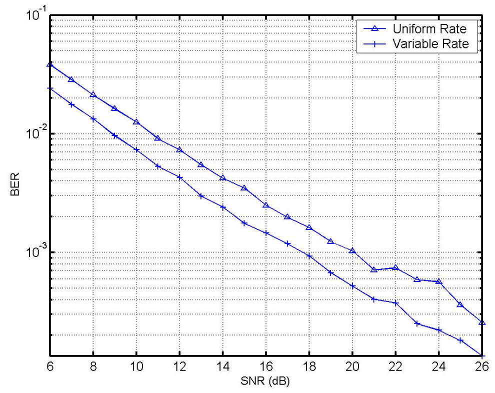
Fig. 5. Cycle counts for different stages of optimization.

The code is optimized with respect to execution time and memory usage. The different optimization stages for encoder, decoder, puncture and insert zeros blocks are shown in Fig. 5.

High levels of optimization are achieved, as shown in Fig. 5. All optimization stages shown include level 3 optimization except for the original code. Results show that for the encoder, a reduction of approximately 63 times in the execution time is achieved by writing the routine in assembly and using loop unrolling as compared to only memory optimization. Similarly, for the decoder, a reduction of approximately two times in the execution time is achieved. Using memory optimization, execution time was reduced approximately 6 and 1.5 times for the puncture and the insert zeros blocks, respectively. The assembly generated by the code composer after memory optimization for the puncture and the insert zeros block was optimized, and no further optimization was achieved by writing these routines in assembly.



(a) 8:1 compression ratio



(b) 32:1 compression ratio by bitstream truncation

Fig. 6. Bit error rate (BER) vs. signal-to-noise ratio (SNR) for embedded foveated image compressed source.

5. SIMULATION

In order to validate the DSP implementation, the entire system is simulated over a Rayleigh fading channel. The performance of this unequal error protection is also compared to that of the uniform error protection case, in which the same rate turbo code is used to protect the entire bitstream. Rate $2/3$ (66.67% transmission efficiency) turbo code is used for the uniform error protection, whereas the equivalent rate for the unequal protection is $3/4$ (75% transmission efficiency). Fig. 6(a) shows the performance characteristics for EFIC, at a compression ratio of 8:1, in terms of bit error rate (BER) vs. signal to noise ratio (SNR) curves. At a BER of 10^{-2}, 1 dB of coding gain is achieved for uniform rate puncturing as compared to the variable rate puncturing. This is so because the former provides larger number of redundant bits and hence offers more error protection.

Fig. 6(b) shows that at a BER of 10^{-2}, 2 dB of coding gain is achieved for variable rate puncturing as compared to the uniform rate puncturing. This is because the former provides greater protection to the more important bits and truncating results in discarding the less protected, less important bits, that contain more errors. Truncating the uniformly protected bitstream does not reduce the BER significantly because the error is uniformly distributed over the entire bitstream.

6. CONCLUSION

In this paper, we presented a real time implementation of the iterative soft output viterbi algorithm based punctured turbo encoder/decoder over a TMS320C6x DSP processor. A speedup of 162x for the encoder and 11.7x for the decoder is achieved over the level three C compiler optimization. All the modules in the system are modelled using computational dataflow models. This is the first publicly available implementation of a SOVA-based turbo decoder on a C6700 DSP processor.

7. REFERENCES

[1] B. Vucetic and J. Yuan, *Turbo Codes*. Kluwer Academic Publishers, 2000.

[2] B. Banister, B. Belzer, and T. Fischer, "Robust image transmission using JPEG2000 and turbo-codes," *IEEE Signal Processing Letters*, vol. 9, pp. 117–119, Apr. 2002.

[3] Z. Wang and A. C. Bovik, "Embedded Foveation Image Coding," *IEEE Transactions on Image Processing*, vol. 10, pp. 1397–1410, Oct. 2001.

[4] C. Berrou and A. Glavieux, "Near optimum error correcting coding and decoding: turbo-codes," *IEEE Transactions on Communications*, vol. 44, pp. 1261 –1271, Oct. 1996.

[5] A. Ushirokawa, T. Okamura, N. Kamiya, and B. Vucetic, "Principles of turbo codes and their applications to mobile communciations," *IEICE Transactions on Fundamentals*, vol. E81-A, pp. 1320–1329, July 1998.

[6] J. Hagenauer, P. Robertson, and L. Papake, "Iterative ('Turbo') decoding of systematic convolutional codes with MAP and SOVA algorithms," *Proc. ITG Conference on Source and Channel Coding, Munich*, Oct. 1994.

[7] S. A. Edwards, *Languages for Embedded Digital Systems*. Kluwer Academic Publishers, 2000.

[8] J. Nikolic-Popovic, "Implementing a MAP Decoder for cdma2000 Turbo Codes on a TMS320C62x DSP Device." Texas Instruments Application Report, SPRA629, May 2000.